2018 IEEE 26th International Conference on Network Protocols

Micro-burst in Data Centers: Observations, Analysis, and Mitigations

Danfeng Shan*‡, Fengyuan Ren*, Peng Cheng*†, Ran Shu*, Chuanxiong Guo+
* Tsinghua University ‡ Xi'an Jiaotong University † Microsoft Research + Bytedance

***Abstract*—Micro-burst traffic is not uncommon in data centers. It can cause packet dropping, which results in serious performance degradation (e.g., Incast problem). However, current solutions that attempt to suppress micro-burst traffic are extrinsic and ad hoc, since they lack the comprehensive and essential understanding of micro-burst's root cause and dynamic behavior. On the other hand, traditional studies focus on traffic burstiness in a single flow, while in data centers micro-burst traffic could occur with highly fan-in communication pattern, and its dynamic behavior is still unclear.**

To this end, in this paper, we re-examine the micro-burst traffic in typical data center scenarios. We find that evolution of micro-burst is determined by both TCP's self-clocking mechanism and bottleneck link. Besides, dynamic behaviors of micro-burst under various scenarios can all be described by the slope of queue length evolution. Our observations also implicate that conventional solutions like absorbing and pacing are ineffective to mitigate micro-burst traffic. Instead, senders need to slow down as soon as possible. Inspired by the findings and insights from experimental observations, we propose S-ECN policy, which is an ECN marking policy leveraging the slope of queue length evolution. Transport protocols utilizing S-ECN policy can suppress the sharp queue length increment by over 2×, and reduce the average query completion time by ∼12-27%.

***Index Terms*—Micro-burst traffic, Packet dropping, TCP, Queue length, Switch buffer.**

I. Introduction

Broadly speaking, micro-burst is highly intense traffic appearing in a relatively short period. Recently, micro-bursts in modern data centers attract much attention in both academic and industry community since they cause serious performance problems [1]–[14]. When massive packets swarm into the same switch port in a twinkling of an eye, the buffer is sharply bulged, and even overwhelmed. The resulting high queuing delays and jitters impose heavy penalties on financial trading applications in data centers [15], and the packet dropping in succession triggers abnormal timeouts, which causes TCP incast throughput collapse [16]–[18], and sluggish response time or low quality of results [19]–[21].

Because of a lack of comprehensive and essential understanding of both root cause and dynamic behavior of micro-burst in data centers, most of existing solutions to suppress micro-burst and eliminate its negative impact are ad hoc and extrinsic, for example, reducing RTOmin [17], limiting the number of concurrent flows and adding jitter in application layer [22], absorbing micro-burst in switch buffer [2], [7], [9], [23]–[26], and smoothing micro-burst through pacing packets at sources [4].

Intuitively, micro-burst traffic stems from various elements, including data generation pattern in applications [5], [27], system call batching and protocol stack processing in OS [5], coalescing packets in NICs [28], [29]. Definitely, the end-to-end congestion control mechanism built-in window-based transport protocol, such as TCP and DCTCP, is the source of a considerable amount of traffic bursts. Historically, lots of work [27], [30]–[35] examined the nature of both micro-burst and macro-burst [34] induced by transport protocols in the context of traditional Internet, and mainly focused on the effect of a single long-lived TCP flow on the formation and dynamic evolution of burst traffic.

The features of modern data center networks are obviously different from the counterparts of traditional Internet, such as symmetrical topology and short network radius. The traffic distribution and pattern are also unique, for example, the short-lived flows are numerous, but a few long-lived flows monopolize the majority of network resources [36]. The new computing paradigms (such as MapReduce [37] and distributed streaming computing [38]) and application systems (such as in-memory computing [39] and distributed machine learning [40]) require various communication patterns, which intensify high fan-in bursty traffic [2]. Moreover, diverse workflow patterns (such as Partition/Aggregate and Dependent/Sequential [41], [42]) further complicate traffic patterns in data centers. Due to the above factors, the conclusions of previous work, which describe the nature of micro-burst and macro-burst induced by a single long TCP flow in Internet, cannot be extended to predicting the dynamic behavior of micro-burst caused by aggregated TCP short flows in data centers, and thus hardly inspire some reasonable guidelines towards removing its negative effects radically.

Considering the features of traffic pattern and distribution in data centers, in this work, we re-examine the micro-burst traffic under the control of window-based transport protocol. The main contributions are summarized in three aspects:

(1) Monitoring the evolution of queue length at fine-grained timescales in typical traffic scenarios and conducting analysis on observations, we infer the temporal and spatial dynamic behaviors of micro-burst, and obtain some interesting findings: 1) The self-clocking mechanism

978-1-5386-6043-0/18/$31.00 ©2018 IEEE
DOI 10.1109/ICNP.2018.00019

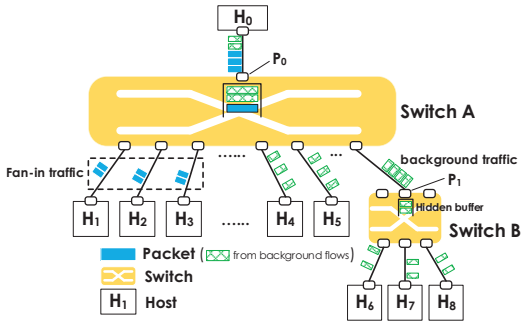


Fig. 1: Micro-burst in data centers

and the bottleneck link jointly dominate the evolution of micro-burst; 2) In any traffic scenarios, regardless of with or without long-lived background flows, and including synchronous or asynchronous fan-in short-lived flows, there is an immutable variable (i.e., the slope of queue length evolution) that can precisely describe the dynamic behaviors of micro-burst in most situations.

(2) Our findings indicate some implications about guidelines towards suppressing micro-burst. The conventional mechanisms, such as absorbing and pacing, are ineffective in the context of data centers. Radically, as for latency-sensitive short message transmission in data centers, the ideal solution is to detect congestion in time and to properly throttle sending rate as soon as possible.

(3) Enlightened by our findings and implications, and leveraging the slope of queue length evolution as an indicator of detecting congestion, we propose an ECN probability marking policy, called S-ECN, to eliminate the negative impact of micro-burst caused by aggregated concurrent flows. Since the slope of queue length can predict the dynamic behavior of micro-burst, when the S-ECN scheme is employed, the queue length increment is effectively suppressed by over 2× compared with DCTCP in data centers. Overall, average query completion time can be reduced by ∼12-27% compared with DCTCP.

The rest of paper is organized as follows. In §II, we present our methodology of studying micro-burst. In §III, we study the micro-burst traffic under different scenarios, and show our experimental observations and findings. In §IV, we propose S-ECN scheme and evaluate its performance. Finally, we discuss some related work in §V, and conclude in §VI.

II. Methodology

In this section, we discuss the scenarios we need to consider, and describe how we observe micro-burst traffic.

A. Scenario Classification

There are various traffic patterns in data center networks, and micro-burst should have different behaviors under different traffic patterns. To comprehensively study the nature and evolution mechanism of micro-burst traffic, five typical scenarios are summarized as follows.

(1) *Synchronous fan-in traffic:* In this scenario, many concurrent flows start simultaneously and are destined for the same target, as depicted in Fig. 1. This scenario is common in many data center application systems. For example, in distributed storage system [16], a client will synchronously read data blocks from lots of storage servers. In user-facing online services employing partition-aggregate structure [19], a parent server will query results from many leaf servers in parallel, and leaf servers will synchronously send responses to the parent server. In memcached systems [39], a web server will communicate with many memcached servers to satisfy a user request.

(2) *Asynchronous fan-in traffic:* Different from the previous scenario, in this scenario, many concurrent flows start asynchronously but are destined for the same target. This scenario exists in multiple data center applications. For example, in distributed machine learning systems [40], lots of servers frequently communicate with a centralized server, and thus communications from different sources can easily overlap although they are asynchronous. And in the MapReduce framework [37], many mappers will transfer intermediate results to reducers, and these transmissions may be asynchronous but can still share the same bottleneck.

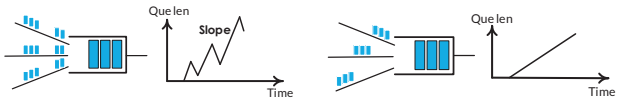
(3) *Fan-in traffic with one background flow:* In this scenario, there is one long-lived background flow before fan-in flows start. The background flow will share the same bottleneck with fan-in flows (congested at port P_0 in Fig. 1).

(4) *Fan-in traffic with several background flows congested at the same hop:* As depicted in Fig. 1, in this scenario, several background flows have already congested at the port P_0 before fan-in flows start. Different from the previous scenario, when fan-in flows start, they can see buffer occupancy caused by background flows at port P_0, which may affect the dynamic behavior of micro-burst traffic.

(5) *Fan-in traffic with several background flows congested at previous hop:* As depicted in Fig. 1, in this scenario, several background flows are congested at port P_1 in the beginning. When fan-in flows arrive at port P_0, they cannot see any buffer occupancy. However, as the congestion point shifts from port P_1 in Switch B to port P_0 in Switch A, the buffer occupancy in the port P_1 will move to the port P_0 in a very short time, which may intensify the traffic burstiness at port P_0.

B. Observing Micro-burst

Capturing or monitoring micro-bursts in data centers is challenging [3], [43], since switches operate at very high speed (10/40Gbps) while micro-bursts occur in very short timescales (hundreds of microseconds). On the other hand, through separately studying individual micro-burst on each link or at end systems, we cannot directly gain the dynamic behavior of micro-burst when lots of flows



(a) Synchronous arrival (b) Asynchronous arrival

Fig. 2: Characteristics of micro-burst can be inferred from queue length evolution.

Resources	Ref. Switch	TPP Switch	S-ECN Switch
Slice Flip-Flops	14158	14777	14700
LUTs	17589	19050	18544

TABLE I: Resource usage in NetFPGA. TPP switch and S-ECN switch are built upon reference switch.

aggregate, which is significant since micro-burst traffic causes problems mainly when fan-in traffic causes fast increasing of queue length.

In this paper, we study micro-burst traffic through observing the evolution of the queue length in the switch, which directly reflects the aggregation behavior of micro-burst traffic. Furthermore, since the queue length in the switch evolves differently under different scenarios, the dynamic behavior of micro-burst caused by aggregation of flows can be inferred from queue length evolution. For example, as shown in Fig. 2, when fan-in flows start synchronously, bursts from different flows may arrive simultaneously. As a result, queue length may fluctuate when it is increasing (Fig. 2(a)). When fan-in flows start asynchronously, bursts from different flows may arrive separately. As a result, queue length may increase in a smoother way (Fig. 2(b)). Moreover, as depicted in Fig. 2(a), the slope of queue length evolution reflects the fan-in degree, since with more concurrent flows the queue length tends to increase faster.

C. Monitoring Queue Length

To observe micro-burst traffic in switch queue, we need to get queue length at fine-grained timescales, because the queue length changes every a few microseconds. Existing techniques in commodity switches can only monitor queue length at a coarse timescale. Inspired by TPP [6], we develop a method that can acquire queue length at per-packet granularity, and introduces little overhead to switch. Specifically, whenever a packet is passing through the switch, we let it carry the timestamp and current queue length in its payload. When the packet arrives at the receiver, the receiver extracts the timestamp and queue length from it, and stores the information into the disk.

We implement the approach on the NetFPGA platform [44]. Specifically, we use a 16-bit register to record the queue length in each port, which is at an 8-byte granularity. We also use a 32-bit register to maintain the timestamp, which is at an 800-ns granularity (i.e., the register increases every 800ns). The timestamp and queue length are put into packets before they enter the output queue, which only needs an additional processing time of 1 clock cycle. Since the clock runs at 125MHz, this implementation only adds 8ns latency to the pipeline. As shown in TABLE I, our implementation (named TPP switch) needs less than 8.3% extra resources compared to the reference switch.

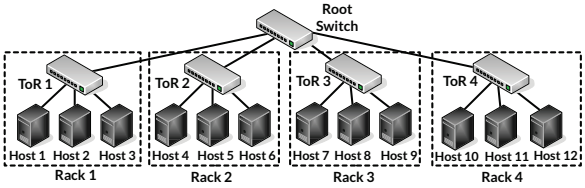


Fig. 3: Network topology in experiments

III. Experimental Observations

In this section, we separately study micro-burst traffic in each scenario listed in §II-A. After that, we present our observations and implications.

A. Testbed

The topology of our testbed is shown in Fig. 3. The testbed contains 12 hosts across 4 racks. Each rack holds 3 hosts connected to a top-of-rack (ToR) switch. All ToR switches are connected through a root switch. Each host is a Dell Optiplex 780 desktop with an Intel® Core™ 2 Duo E7500 2930 MHz CPU, 4 GB memory, a 500GB hard disk, and an Intel® 82567LM Gigabit Ethernet NIC, running CentOS 5.11 with GNU/Linux kernel 2.6.38. All switches are NetFPGA cards with four Gigabit Ethernet networking ports. Buffer size in each output port can be arbitrarily set between 1KB and 512KB. The round-trip time (RTT) between hosts from different racks is about 50μs without queueing. The queue length in root switch is monitored.

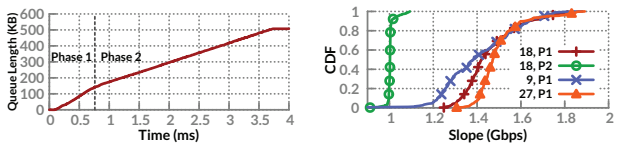
B. Observations and Analysis

(1). Synchronous fan-in traffic

Experiment settings: Firstly, we let Host 10 send queries to Host 1-9 at the same time and these hosts will simultaneously send a response message to Host 10. Each host is used to emulate multiple senders. Query size is 50B and response message size is 1000KB. The buffer size in each output port is set to 512KB. End hosts use TCP NewReno as their transport protocols.[1] Large Send Offload is disabled, but it will not affect our main results (see discussion in §III-E). Delayed ACK is disabled as is suggested in [45]. We use default values for other TCP parameters.

Experiment results: Fig. 4(a) depicts the queue length evolution in the congested port when there are 18 senders.

[1]We have studied different TCP variants (Reno, SACK, CUBIC) and observed the similar results.



(a) Queue length (b) Distribution of slope

Fig. 4: Experiment results in the synchronous fan-in scenario. In (b), "18, P1" stands for slope in Phase 1 with 18 fan-in flows

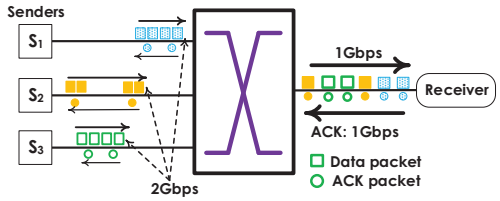


Fig. 5: Explanation of Phase 2

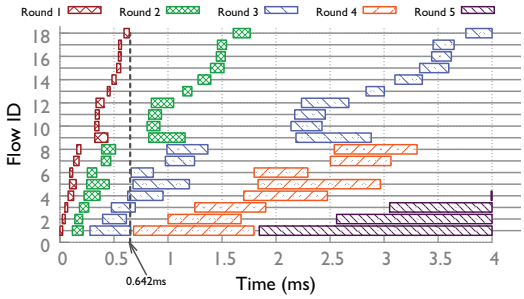


Fig. 6: duration of transmitting each round of packets

We also repeat the experiment 100 times and calculate the slope of queue length evolution in each experiment. The slope distributions are shown in Fig. 4(b). We have three observations.

First, as shown in Fig. 4(a), the queue length evolution has two phases: sharply increasing for a relatively short period at the beginning (Phase 1) and relatively slowly increasing after that (Phase 2). Particularly, in Phase 2, the queue length increases at a constant rate, without any impulses or jitters.

Second, as shown in Fig. 4(b), in Phase 2, *the queue length is not only increasing at a constant rate but the increasing rate is always 1Gbps*. Specifically, when there are 18 flows, the average slope is 1.005Gbps, and the coefficient of variation is only 0.018. We find that this phenomenon is caused by TCP's self-clocking system and evolution of congestion window in slow start phase, as depicted in Fig. 5. Specifically, since data packets are received at 1Gbps, the receiver is always acknowledging data at 1Gbps, which is the speed of the congested port. On the other hand, since all flows are in slow start phase, each sender introduces two data packets into the network when receiving an ACK packet. As a result, the total generated traffic from all senders is 2Gbps. Thus, the overall queue length increasing rate is 1Gbps. Furthermore, at packet granularity, ACK packets are evenly spread across bottleneck link. Since the topology is symmetric, after these ACKs arrive at senders, they will evenly trigger transmissions of data packets, and thus data packets are evenly spread across input links. As a result, after these packets aggregate, the queue length in congested port is smoothly increasing at a constant rate.

Third, as shown in Fig. 4(b), in Phase 1, the slope is much larger than that in Phase 2; and with more concurrent flows, the slope in Phase 1 tends to be larger. This is because sources are sending the first round of packets, and the self-clocking system does not take effect in this phase. To prove this, we find the duration of transmitting each round of packets in each flow, which is shown in Fig. 6. The latest ending time of transmitting the first round of packets is at 0.642ms, which is in accordance with the ending time of Phase 1 in Fig. 4(a).

(2). Asynchronous fan-in traffic

Experiment settings: In this experiment, 18 flows randomly start during a period of 2ms, which are from Host 1-9 to Host 10. Other settings keep unchanged.

Experiment results: The queue length evolution is shown in Fig. 7. As the first rounds of packets from different senders asynchronously arrive at the congested port, in Phase 1 the queue length does not increase as sharply as that in previous scenario where these packets arrive synchronously. However, in Phase 2, the queue length is smoothly increasing at a constant rate of port speed, which is similar to that in the synchronous scenario. This is because all flows have finished sending the first round of packets, and all packet departures in this phase are triggered by ACKs. In addition, as in previous scenario, ACKs are evenly acknowledging data at 1Gbps, and each ACK packet triggers transmissions of two packets since flows are in slow start phase. Therefore, traffic arriving rate at the congested port is 2Gbps and slope is 1Gbps.

(3). Fan-in traffic with one background flow

Experiment settings: Along with query traffic, we add one background flow. Specifically, at the beginning, Host 9 begins to send data to Host 11 as fast as it can. After 0.5 seconds, Host 10 begins to query Host 1-8. Other settings are not changed.

Experiment results: The experiment is repeated 100 times. As shown in Fig. 8, the slope in Phase 1 is larger than the port speed as in previous scenarios, but the slope in Phase 2 becomes much smaller than the port speed. This is because the congestion window of the background flow has exceeded the slow start threshold by the time other fan-in flows start, and thus the background flow introduces less than two data packets into the network when receiving an ACK packet. On the other hand, the other flows inject two data packets when receiving an

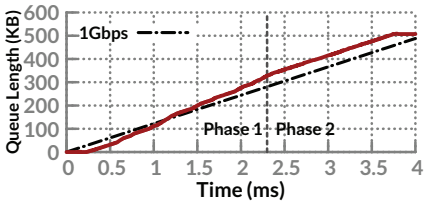


Fig. 7: Results in the asynchronous fan-in scenario: queue length evolution (left) and slope distribution (right).

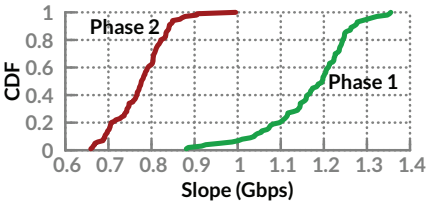


Fig. 8: Slope distribution with one background flow.

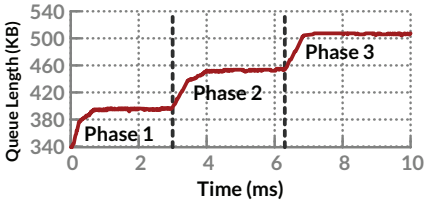


Fig. 9: Experiment results when there are several background flows congested at the same hop: queue length evolution (left) and slope distribution (right). "P2" stands for "Phase 2".

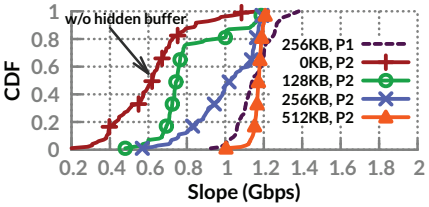


Fig. 10: Slope distribution when there is hidden buffer in the previous hop.

ACK. Overall, the arriving rate of total traffic is less than 2Gbps, and thus the slope is smaller than 1Gbps.

(4). Fan-in traffic with several background flows congested at the same hop

Experiment settings: Along with query traffic, we add three background flows, which are from Host 1, 4, 7 to Host 11 and Host 12. After 0.5 seconds, Host 10 begins to query hosts in other racks (excluding Host 1, 4, 7) and these hosts send a response message to Host 10.

Experiment results: The queue length evolution is shown in Fig. 9. The queue length evolution can be split into multiple phases. Before the fan-in flows start, a fraction of buffer in the root switch is already occupied by background flows. In Phase 1, the first round of packets from the fan-in flows arrive at the port simultaneously, therefore the queue length experiences a sharp increment. After that, since there is buffer occupancy caused by background flows before the fan-in flows start, the packets from fan-in flows are queued behind those packets from background flows, and fan-in flows will not inject new packets into the network for a while. As a result, the queue length stops sharply increasing. Similarly, in Phase 2, the queue length increases sharply at the beginning, which is caused by the second round of packets, and the queue length stops sharply increasing after that. Finally, fast queue length increasing in Phase 3 is caused by the third round of packets.

The distribution of slope in each phase is shown in Fig. 9. We have three main observations. *First,* we find that the slope after Phase 1 (i.e., Phase 2 and Phase 3) is much smaller than that without background flows. This is because some of the arriving packets in these phases are from background flows, and these background flows are in the congestion avoidance phase, in which the congestion window is increased by one packet every round trip time. Therefore, although ACKs are acknowledging data at 1Gbps, data packets are sent at a rate smaller than 2Gbps. *Second,* after Phase 1, the slope in the later phase (e.g., Phase 3) is larger than that in the earlier phase (e.g., Phase 2). This is because ACKs from slow start flows introduce more packets into the network (each ACK packet introduces 2 data packets) than those from flows in the congestion avoidance phase (each ACK packet introduces *less than* 2 data packets), and thus the fraction of packets from slow start flows becomes increasingly large. As a result, in the reverse path, the ratio of ACKs from slow start flows also becomes larger. Finally, after these ACKs arrive at the senders, the total packet departure rate is larger in the later phase, and the slope becomes larger in the later phase. *Third,* the slope distributions are similar when the number of background flows varies. For example, the average slopes are (0.613Gbps, 0.622Gbps, 0.641Gbps) when there are (3, 6, 9) background flows. This is because the ratio of packets from slow start phase changes little as the number of background flows varies, and the congestion window increasing rate from background flows can be ignored compared to that from slow start flows.

(5). Fan-in traffic with several background flows congested at previous hop

Experiment settings: In this experiment, the background flows are from Host 7, 8, 9 to Host 11 and Host 12, thus, these background flows are congested in ToR switch 3. After 0.5 seconds, Host 10 begins to query Host 1-6 and they send a response message to Host 10. At this time, the congestion point will shift to the next hop in the root switch. Other experiment settings remain unchanged.

Experiment results: Fig. 10 shows the distribution of slope. In this figure, we compare the slope distribution

with that in the previous experiment in which the congestion point does not change. We observe that in this scenario the slope in Phase 2 is much larger than that without shift of congestion point, and may even be larger than the port speed. This phenomenon is caused by the fast shift of buffer occupancy in the previous hop. Specifically, in Phase 2, arriving packets at the root switch are from fan-in flows and background flows. The packet transmissions from fan-in flows follow self-clocking mechanism. However, the transmissions of packets from background flows are not determined by self-clocking mechanism, since they are from buildup queue in the previous hop (i.e., ToR switch 3), and they are sent at the port speed of ToR switch 3. As a result, the queue length increasing is intensified by these packets. Since the buffer occupancy in previous hop cannot be seen by the fan-in flows at the beginning, we call it *hidden buffer*.

We further examine the effect of hidden buffer by setting the buffer size of ToR switch 3 to 128KB, 256KB, and 512KB, respectively. As shown in Fig. 10, the slope increases as the number of packets in hidden buffer increases. Specifically, when buffer sizes are 128KB, 256KB, and 512KB, the average slopes are 0.804Gbps, 1.009Gbps, and 1.170Gbps, respectively.

Although the queue length increasing is intensified by hidden buffer, the slope in Phase 2 has an upper bound of $2aR/(a+R)$, where a is the arriving rate of first round packets from the fan-in flows and R is the port speed. This is because after 1st round packets arrive at the congested port, the fraction of bandwidth occupied by fan-in flows is about $aR/(a+R)$. Therefore, in the next round, these fan-in flows will generate traffic at $2aR/(a+R)$ in total. Meanwhile, sending rate of traffic from the hidden buffer to the congested port is at most R. Therefore, the queue length increasing rate is no larger than $2aR/(a+R)$.

C. Summary

In summary, when multiple flows aggregate in a queue, the queue length evolution can be divided into two phases. In Phase 1, queue length increasing is caused by the arriving of 1st round packets. The slope in Phase 1 directly reflects the communication pattern and intensity (i.e., fan-in degree). Specifically, when communications are synchronous, or more concurrent flows aggregate in a queue, the slope in this phase is larger.

In Phase 2, the arriving traffic rate is limited by the bottleneck speed, because the total amount of sent data among all bottlenecked flows is always acknowledged at bottleneck rate. Besides, the traffic arriving rate is directly reflected by the slope of queue length evolution. Under different traffic scenarios, dynamics of slope can be summarized as following three laws:

Law 1: Without background flows, the slope in Phase 2 is equal to the port speed.

Law 2: When there is one background flow, or several background flows are congested at the same hop, the slope in Phase 2 is smaller than the port speed.

Law 3: When several background flows are congested at previous hop, slope in Phase 2 might be larger than port speed, but no larger than $2aR/(a+R)$, where a is the arriving rate of 1st round packets and R is the port speed. In particular, the slope should be no larger than $2R$ (when $a \to +\infty$).

Moreover, from a microscopic perspective, packet arrival pattern in forward path is evenly scheduled by ACK packets in reverse path. Specifically, data packets from different sources are sent at full rate to bottleneck link. After they arrive at the receiver, the ACKs will evenly spread in the reverse path of bottleneck link. After these ACKs are fanned out to multiple senders, they will evenly trigger the transmissions of data packets into the network. As a result, in the bottleneck queue, the queue length increases smoothly.

D. Implications

These experimental observations have some implications:

(1) Slope is an important indicator to mitigate micro-burst traffic. Slope directly reflects the bursty degree of aggregated traffic. Specifically, the slope in Phase 1 reflects the flow concurrency, and the slope in Phase 2 reflects the mismatch between congestion window increasing rate and bottleneck rate. By carefully slowing down senders according to the slope, the sharp queue length increasing caused by micro-burst traffic can be effectively suppressed.

(2) TCP pacing cannot help ease the sharp queue length increasing caused by fan-in traffic. In the literature [30], [32], [46], [47], TCP pacing is considered as an effective method to smooth traffic burstiness in a single flow by evenly spreading a window of packets over a round-trip time. However, in data centers with fan-in traffic, traffic burstiness comes from the aggregation of multiple flows, and transmissions of data packets have already been evenly triggered by ACK packets (as is analyzed in §III-B). Thus, TCP pacing has little effect on queue length evolution, and may not alleviate the impairments caused by micro-burst traffic. To validate this, we conduct simulations on ns-2 platform [48], [49]. The results are shown in Fig. 11.

(3) Simply absorbing micro-burst traffic may not help avoid packet dropping, but partly magnifies micro-burst traffic. Senders won't slow down before they receive congestion signals. If packets are absorbed without notifying senders of congestion, the senders will believe that network is still fine and introduce more packets into the network. Specifically, if the sender is in slow start phase, absorbing every packet will introduce another two packets into the network.

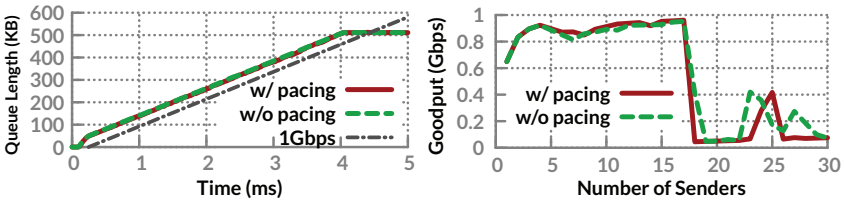


Fig. 11: Performance of TCP pacing[2]: queue length evolution (left) and Incast performance (right).

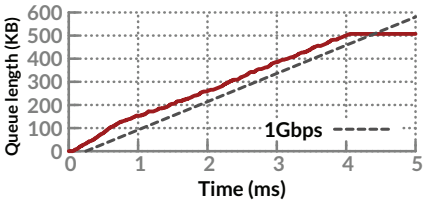


Fig. 12: Queue length evolution when TSO is turned on.

E. Discussion

Impact of other micro-burst traffic. One might ask whether other traffic burstiness (e.g., caused by TCP Segment Offload) affects our observations. We find that these kinds of micro-burst traffic only cause queue jitter at very small timescales, because they disrupt the even transmission of data packets. However, the main observations are not affected. For example, we conduct the same experiment as that in §III-B, and use the same experiment settings except that TSO is turned on. As shown in Fig. 12, the queue length evolution is similar to that without TSO.

Are these observations specific to TCP? Most of our phenomena can also be observed in other TCP-like protocols, because they use the same self-clocking mechanism and similar congestion window adjustment algorithm as TCP. Besides, the hidden buffer is caused by persistent buffer occupancy from long flows. Since most transport protocols can not promise zero buffer occupancy, hidden buffer also exists in other protocols.

IV. Mitigating Micro-burst

From previous experiments, we find that to mitigate micro-burst traffic, senders need to slow down in time. In this section, we discuss the existing solutions that can potentially mitigate micro-burst traffic. After that, inspired by our implications, we show how to use the slope to mitigate micro-bursts.

A. Existing Solutions

ECN: Explicit Congestion Notification (ECN) [50] can explicitly notify senders of congestion before buffer overflows. Therefore, by using ECN, packet dropping caused by micro-burst traffic may be avoided. In data centers, ECN is used by lots of transport protocols, such as [4], [19], [21], [51], [52]. In all protocols, packets are marked with ECN when the queue length is larger than the marking threshold.

However, queue-length-based ECN marking has limitations when micro-burst traffic causes sharp queue length increasing, because packets are not marked until the queue length reaches a threshold. After queue length reaches the threshold, it takes time for all senders to receive the feedbacks from switch queue. During this time, the queue length still keeps sharply increasing and buffer may overflow before all senders slow down.

Other Protocols: Besides ECN, micro-burst may be mitigated by replacing TCP-like protocols with other protocols. For example, micro-burst may be mitigated by carefully scheduling flows [53]–[55] or packets [56], using delay as congestion signal [57], [58], or avoiding packet dropping with lossless network [52]. However, in this paper, we focus on using our implications to improve current TCP-like protocols. Investigation of micro-burst with other protocols is part of our future work.

B. Design of S-ECN

To effectively suppress the micro-burst traffic, we propose S-ECN — a slope-based ECN marking scheme in switches. S-ECN is inspired by the implication of slope: slope reflects the mismatch between the congestion window increasing rate and bottleneck rate, thus sharp queue increasing can be suppressed by properly marking packets according to slope. Specifically, we mark packets immediately when the queue length begins to increase. Furthermore, the fraction of marked packets indicates how large the slope is: the bigger the slope, the larger the fraction.

Specifically, when slope (denoted by s) is equal to or even larger than the port speed (denoted by R, which is constant in general), then flows are likely in slow start phase; all packets are marked with ECN so that flows will slow down immediately. When slope is lower than the port's speed, then packets are marked at a probability of s/R. Finally, when slope is lower than or equal to 0 (i.e., the queue length is not increasing), none of the packets are marked. The marking probability (denoted by $Prob$) as a function of slope s is given by equation (1).

$$Prob = \begin{cases} 0, & s \leqslant 0, \\ \frac{s}{R}, & 0 < s < R, \\ 1, & s \geqslant R \end{cases} \tag{1}$$

C. Implementation of S-ECN

We implement our scheme on NetFPGA platform [44]. For each packet arriving at the output queue, we get the following three values:

(1) P: Total size (in bytes) of the arriving packet

(2) I: Time interval (in clocks, 1 clock = 8ns) between arrivals of the previous packet and the current packet

[2]Simulation topology: 40 servers are connected through the same switch with 128KB buffer. Workload: (a) The same as that in §III-B. (b) Each sender sends 64KB data to the same receiver.

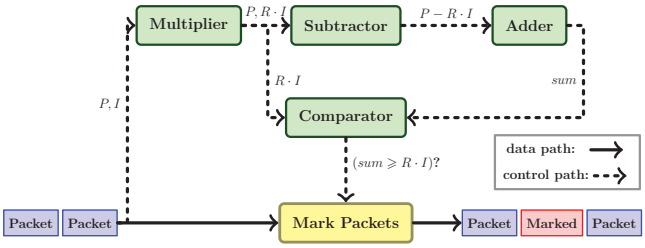


Fig. 13: Structure of S-ECN

(3) R: Port speed (in Gbps)

We next show how to mark packets based on these values. When a packet arrives at the switch's output port, the current traffic arriving rate to the output port can be given by P/I. Then the current slope is denoted by $s = P/I - R$. If $0 < s < R$, the marking probability is $Prob = s/R = \frac{P-R \cdot I}{R \cdot I}$. Finally, equation (1) can be rewritten as

$$Prob = \begin{cases} 0, & P \leqslant R \cdot I, \\ \frac{P-R \cdot I}{R \cdot I}, & R \cdot I < P < 2R \cdot I, \\ 1, & P \geqslant 2R \cdot I \end{cases} \quad (2)$$

To implement S-ECN, we need to implement probability marking. One method to implement probability marking is by random number generator. However, using a random number generator will introduce large overhead in hardware. Therefore we use another method: instead of marking with probability $Prob$, we can mark every $1/Prob$ packets. Specifically, we can set up a threshold $T = \frac{R \cdot I}{P-R \cdot I}$ and there is a counter counting the number of unmarked packets. When the number of unmarked packets is equal or larger than T, then the next packet will be marked with ECN.

However, calculating the threshold needs to use a divider, which will also introduce overhead. Fortunately, we can replace divider with adder and comparator. Specifically, let K be the number of unmarked packets, then an arriving packet is marked if $K \geqslant \frac{RI}{P-RI}$. The condition can be rewritten as $K \cdot (P - RI) \geqslant RI$. Instead of counting unmarked packets, we can add up $(P - RI)$ for every arriving packet, and compare the sum with RI. In other words, the arriving packet is marked with ECN if $\sum_{j=1}^{K}(P_j - RI_j) > RI_K$.

In summary, our implementation structure is shown in Fig. 13. When a packet arrives, we calculate $R \cdot I$ and $(P - R \cdot I)$. Then $\sum_{j=1}^{K}(P_j - RI_j)$ is calculated by add $(P - R \cdot I)$ to accumulated sum (i.e., sum = sum + $P - RI$). Finally, the ECN marking decision is made by comparing $R \cdot I$ with the sum. TABLE I shows that our implementation needs less than 6% extra resources in NetFPGA.

D. Evaluation of S-ECN

In this section, we use realistic experiments (in the testbed described in §III-A) to show the performance of the S-ECN scheme.

(1). Protocols compared

S-ECN scheme is only an ECN marking scheme in the switch. It can coordinate with different end host algorithms. However, since the slope of queue length is indicated by the fraction of marked packets, S-ECN can work better if end hosts decrease their sending rate according to the fraction of marked packets. Therefore, during evaluation, we use DCTCP as end host algorithms in that DCTCP cuts the congestion window according to the fraction of marked packets.

Specifically, the protocols we consider and parameter settings are listed as follows:

(i) DCTCP: The protocol proposed in [19]. We set $g = 0.125$. In the switch, ECN marking is based on instantaneous queue length. The ECN threshold (denoted by K) is set to 32KB.

(ii) DCTCP+S-ECN: End hosts use DCTCP as their congestion control algorithms. We set $g = 0.125$. The switch marks packets according to slope (as described in §IV-B) when queue length is smaller than ECN threshold, and marks all packets when queue length is larger than ECN threshold.

(2). Microbenchmarks

Suppression of sharp queue increasing: First, we show how our scheme can help suppress the sharp queue length increasing. In the experiment, we set the buffer size to 512KB. We let a client (Host 10) query other servers (Host 1-9) for 20MB data and get the instantaneous queue length in the congested port.

The queue length evolution of each protocol is shown in Fig. 14. With DCTCP, the queue length can sharply increase to 104KB at the beginning, which is over 3× higher than ECN threshold (32KB). With DCTCP+S-ECN the queue length can be controlled under 47KB, which is reduced by over 2× compared with DCTCP.

One may wonder whether lowering ECN threshold can also effectively suppress queue increasing, because senders will slow down earlier. To show this, we set the ECN threshold to 2KB, which is the lowest value such that there is no throughput loss (i.e., buffer does not underflow). As shown in Fig. 14 (dashed line), with DCTCP, the queue length can still increase to 65.7KB. As expected, this is smaller than the queue length increment with ECN threshold set to 32KB, but the increment is larger than that using DCTCP+S-ECN protocol. This is because some early started senders do not slow down when the queue length is lower than ECN threshold, and they will introduce two times more traffic at the next RTT.

We further show the queue length evolution when dequeue marking [51] is used. With dequeue marking (i.e., packets are marked when dequeued rather than enqueued), the senders will receive congestion signal earlier. As shown in Fig. 14, the queue length increment can be reduced to 74KB. However, it is larger than that with DCTCP+S-ECN. This is because it takes at least one RTT for all

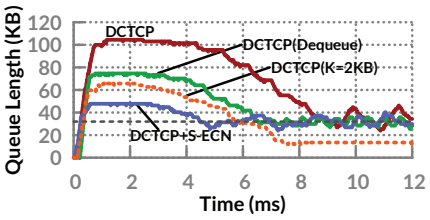


Fig. 14: Queue length evolution with various protocols

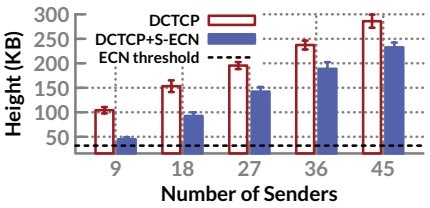


Fig. 15: Maximum height of queue length with more senders

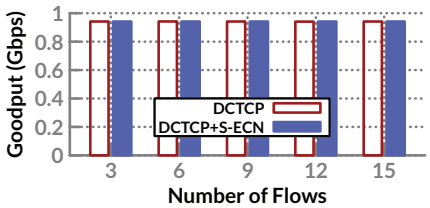


Fig. 16: Goodput when the number of flows is small

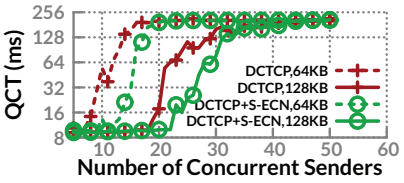
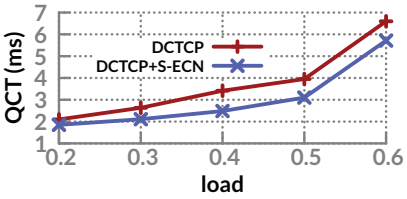
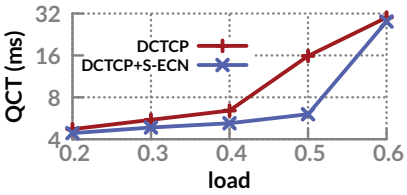


Fig. 17: Query Completion Time in incast scenario



(a) Average



(b) 99th percentile

Fig. 18: Query completion time with benchmark traffic. Note that in (b) y axis is log-scaled.

senders to slow down, while S-ECN marks packets as soon as queue length begins to increase.

To show the maximum height the queue length can reach when there are more concurrent flows, we enlarge the number of flows by letting each sending host (i.e., Host 1-9) emulate multiple senders. We re-conduct the previous experiment when the number of slaves is 9, 18, 27, 36, and 45. Fig. 15 shows the maximum height of each protocol. Protocols using S-ECN scheme in the switch can well control the queue length when the number of concurrent flows is large. For example, when there 36 senders, with DCTCP+S-ECN the maximum queue length can be reduced by 20.7% compared with DCTCP.

Network utilization: In this part, we show the network utilization when the number of flows is small. We start 3 flows every 1 second, which are from different senders and destine for the same receiver. Each flow will keep sending data after it starts.

We obtain the total throughput among all flows when a new bunch of flows start, which is shown in Fig. 16. Throughput of DCTCP+S-ECN is always over 900Mbps. Thus, the link is fully utilized. For example, when there are 3 flows, the throughput of DCTCP and DCTCP-S-ECN is 941.4Mbps and 941.5Mbps, respectively.

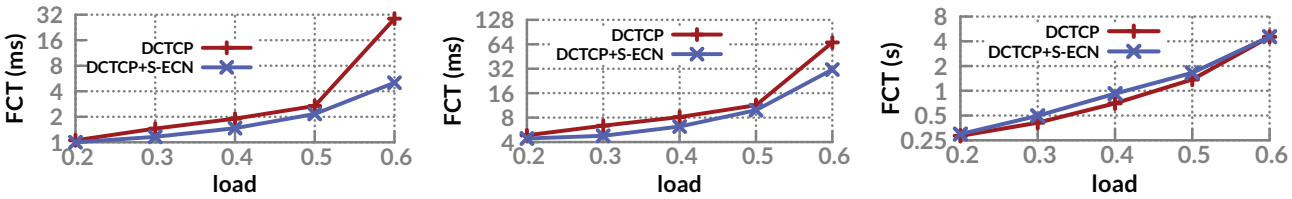
Incast performance: In this part, we'll show how S-ECN scheme improves incast performance. In this experiment, a client (Host 10) will send a query to several servers (Host 1-9) simultaneously; each server will respond with 1MB/n data, where n is the number of servers. The client waits until it receives all data. Each host is used to emulate multiple senders [20]. We repeat each experiment 100 times and Fig. 17 shows the query completion time for each protocol. When the buffer size is 128KB, DCTCP+S-ECN performs much better than DCTCP when the number of senders is between 19 and 32. Similar results can be obtained when buffer size is 64KB (dashed line).

(3). Benchmark traffic

Experiment settings: We use a realistic workload to evaluate our scheme, which is derived from a data center supporting web search service [19]. In the workload, there are two kinds of traffic between servers. One is query traffic, where a client will periodically send a query to all other servers and each server will send a response message back. The total response size is 100KB. The other kind of traffic is background traffic, which follows a one-to-one pattern. The flow size distribution is from [19]. Arrivals of query and background flows follow a Poisson process. The ratio of query traffic and background traffic is also drawn from [19]. In our experiments, switch buffer of each port is 128KB, and RTO_{min} is set to 10ms [17], [59]. Each experiment lasts for 5 minutes. Over 350,000 query flows and 140,000 background flows are generated, respectively.

Experiment results: Fig. 18 shows the query completion time. By suppressing the fast queue increasing, flows can finish faster by suffering fewer timeouts. More precisely, compared to DCTCP protocol, DCTCP+S-ECN can reduce the average query completion time by ∼12%-27%, and 99^{th} query completion time by ∼6%-62%.

Fig. 19 shows the flow completion times (FCTs) of different sizes of background flows. By suppressing the fast queue increasing, small background flows can also finish faster, especially when network load is larger and the flow concurrency is higher. Specifically, as shown in Fig. 19(a), compared with DCTCP, DCTCP+S-ECN can reduce the average FCT for small background flows by 5.1% (load=0.2), ∼20% (load=0.3, 0.4, 0.5), and 82.5% (load=0.6), respectively. As shown in Fig. 19(b), the 99^{th} percentile FCT can be reduced by 4.2% (load=0.2), 23.4% (load=0.3), 25.3% (load=0.4), 18.9% (load=0.5), and 11.3% (load=0.6), respectively. However, for large background flows in $(10\text{MB}, \infty)$, DCTCP+S-ECN is slightly



(a) $(0, 100\text{KB}]$: Average (b) $(0, 100\text{KB}]$: 99th percentile (c) $(10\text{MB}, \infty)$: Average

Fig. 19: Flow completion time of background traffic. Note that y axis is log-scaled.

worse than DCTCP. This is because S-ECN does not distinguish traffic bursts caused by congestion control from bursts caused by other miscellaneous reasons (e.g., system batching) and may mistakenly mark some packets. Thus, further improving S-ECN scheme is part of our future work.

V. Related Work

Lots of studies investigate the burstiness in wide area networks. Zhang et al. [30] find that ACK packets can be bunched up when encountering queuing, which in turn causes bursty transmission at sender. The phenomenon is called ACK-compression. Bennett et al. [60] find that ACK reordering can result in traffic burstiness. Aggarwal et al. [46] outline the causes of burstiness in TCP and investigate the TCP pacing's performance. Jiang and Dovrolis [27] identify several causes of bursts from individual flows and examine their effects. They [32] also find that TCP's self-clocking and network queuing can shape the packet interarrivals into a two-level ON-OFF pattern. In [35], Shakkottai et al. studied the burstiness of TCP flows at packet level. By analyzing real network traces, Blanton and Allman [33] find that large bursts can always cause packet loss but they rarely occur. Several burst mitigation methods are studied in [34]. These studies mainly focus on burstiness in a single flow, while we study the behavior of micro-burst traffic when many flows aggregate in data center networks.

In data centers, Benson et al. [61] analyze the traces from ten data centers, and find that the traffic exhibits an ON/OFF pattern. Zhang et al. [11] examine the microsecond-level behaviors of the traffic in a production data center and find that most of the congestion events are caused by the micro-burst traffic. Lu et al. [25] propose to use large off-chip packet buffer to absorb traffic bursts. Kapoor et al. [5] study the new causes of bursts, including offloading features in NIC, batching schemes, and bursty system calls. In [8], [62], Ghorbani et al. find that short-lived congestion caused by micro-burst brings challenges to load balancing systems. Several tools [3], [6], [12], [13], [63], [64] are developed to detect micro-bursts. Different from these studies, we focus on the nature of micro-burst traffic introduced by the transport protocol.

VI. Conclusion

In this paper, we use real experiments to examine the micro-burst traffic through observing queue length at fine-grained timescale in typical traffic scenarios. We find that self-clocking mechanism and bottleneck link jointly dominate the evolution of micro-burst. We also find that slope of queue length evolution can describe the dynamic behavior of micro-burst under all scenarios. These findings implicate that traditional solutions like absorbing and pacing to mitigate micro-burst traffic are not effective in data centers. To suppress the micro-burst traffic, the sending rate needs to be throttled as soon as possible. Enlightened by our findings and implications, we propose S-ECN policy, which takes advantage of the slope of queue length evolution and randomly marks packets with ECN. The experiment results show that protocols using S-ECN policy can effectively mitigate micro-burst traffic.

Acknowledgements. The authors gratefully acknowledge the anonymous reviewers for their constructive comments. We would also like to thank Songwu Lu for the valuable suggestions on the paper's initial version and Tong Zhang for the proof reading.

References

[1] Arista Networks, "Myths about "Microbursts"," White Paper, 2009. [Online]. Available: http://www.arista.com/assets/data/pdf/7148sx-ixnetwork-microburst.pdf

[2] ——, "Microbursts, Jitter and Buffers," White Paper, 2014. [Online]. Available: https://www.arista.com/assets/data/pdf/TechBulletins/AristaMicrobursts.pdf

[3] F. Uyeda, L. Foschini, F. Baker, S. Suri, and G. Varghese, "Efficiently Measuring Bandwidth at All Time Scales," in *NSDI 2011*.

[4] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center," in *NSDI 2012*.

[5] R. Kapoor, A. C. Snoeren, G. M. Voelker, and G. Porter, "Bullet Trains: A Study of NIC Burst Behavior at Microsecond Timescales," in *CoNEXT 2013*.

[6] V. Jeyakumar, M. Alizadeh, Y. Geng, C. Kim, and D. Mazières, "Millions of Little Minions: Using Packets for Low Latency Network Programming and Visibility," in *SIGCOMM 2014*.

[7] D. Shan, W. Jiang, and F. Ren, "Absorbing micro-burst traffic by enhancing dynamic threshold policy of data center switches," in *INFOCOM 2015*.

[8] S. Ghorbani, B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Micro Load Balancing in Data Centers with DRILL," in *HotNets 2015*.

[9] D. Shan, W. Jiang, and F. Ren, "Analyzing and Enhancing Dynamic Threshold Policy of Data Center Switches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2454–2470, 2017.

[10] D. Shan and F. Ren, "Improving ECN Marking Scheme with Micro-burst Traffic in Data Center Networks," in *INFOCOM 2017*.

[11] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution Measurement of Data Center Microbursts," in *IMC 2017*.

[12] R. Joshi, T. Qu, M. C. Chan, B. Leong, and B. T. Loo, "BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks," in *APSys 2018*.

[13] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, and O. Rottenstreich, "Catching the Microburst Culprits with Snappy," in *SelfDN 2018*.

[14] D. Shan and F. Ren, "ECN Marking with Micro-burst Traffic: Problem, Analysis, and Improvement ," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2018.

[15] "Performance Management for Latency-Intolerant Financial Trading Networks." Financial Service Technology.

[16] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems." in *FAST 2008*.

[17] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication," in *SIGCOMM 2009*.

[18] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP Incast Throughput Collapse in Datacenter Networks," in *WREN 2009*.

[19] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *SIGCOMM 2010*.

[20] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better Never Than Late: Meeting Deadlines in Datacenter Networks," in *SIGCOMM 2011*.

[21] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware Datacenter TCP (D2TCP)," in *SIGCOMM 2012*.

[22] J. Rothschild, "High performance at massive scale: Lessons learned at facebook," https://www.youtube.com/watch?v=KIRzk08NMNo, 2009.

[23] S. Das and R. Sankar, "Broadcom smart-buffer technology in data center switches for cost-effective performance scaling of cloud applications," *Broadcom White Paper*, 2012.

[24] Extreme Networks, "Congestion Management and Buffering in Data Center Networks," White Paper, 2014.

[25] G. Lu, R. Miao, Y. Xiong, and C. Guo, "Using CPU As a Traffic Co-processing Unit in Commodity Switches," in *HotSDN 2012*.

[26] A. Singh *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *SIGCOMM 2015*.

[27] H. Jiang and C. Dovrolis, "Source-level IP Packet Bursts: Causes and Effects," in *IMC 2003*.

[28] S. Makineni, R. Iyer *et al.*, "Receive Side Coalescing for Accelerating TCP/IP Processing," in *HiPC 2006*.

[29] K. Christensen, P. Reviriego, B. Nordman, M. Bennett, M. Mostowfi, and J. Maestro, "IEEE 802.3az: the road to energy efficient ethernet," *Communications Magazine, IEEE*, vol. 48, no. 11, pp. 50–56, November 2010.

[30] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-way Traffic," in *SIGCOMM 1991*.

[31] W. chun Feng, P. Tinnakornsrisuphap, and I. Philip, "On the burstiness of the tcp congestion-control mechanism in a distributed computing system," in *ICDCS 2000*.

[32] H. Jiang and C. Dovrolis, "Why is the Internet Traffic Bursty in Short Time Scales?" in *SIGMETRICS 2005*.

[33] E. Blanton and M. Allman, "On the Impact of Bursting on TCP Performance," in *PAM 2005*.

[34] M. Allman and E. Blanton, "Notes on Burst Mitigation for Transport Protocols," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, Apr. 2005.

[35] S. Shakkottai, N. Brownlee, and k. claffy, "A Study of Burstiness in TCP Flows," in *PAM 2005*.

[36] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Data Center Traffic: Measurements & Analysis," in *IMC 2009*.

[37] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[38] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed Stream Computing Platform," in *ICDMW 2010*.

[39] R. Nishtala *et al.*, "Scaling Memcache at Facebook," in *NSDI 2013*.

[40] L. Mai, C. Hong, and P. Costa, "Optimizing network performance in distributed machine learning," in *HotCloud 2015*.

[41] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks," in *SIGCOMM 2012*.

[42] R. Kapoor, G. Porter, M. Tewari, G. M. Voelker, and A. Vahdat, "Chronos: Predictable Low Latency for Data Center Applications," in *SoCC 2012*.

[43] Cisco Systems, Inc., "Monitor Microbursts on Cisco Nexus 5600 Platform and Cisco Nexus 6000 Series Switches," White Paper, October 2014.

[44] "The NetFPGA Project," http://netfpga.org/.

[45] M. Yu, A. Greenberg, D. Maltz, J. Rexford, L. Yuan, S. Kandula, and C. Kim, "Profiling Network Performance for Multi-tier Data Center Applications," in *NSDI 2011*.

[46] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of TCP pacing," in *INFOCOM 2000*.

[47] D. Wei, P. Cao, and S. Low, "TCP Pacing Revisited," in *INFOCOM 2006*.

[48] "ns-2," http://www.isi.edu/nsnam/ns/.

[49] "A TCP pacing implementation for NS2," http://netlab.caltech.edu/projects/ns2tcplinux/ns2pacing/.

[50] K. Ramakrishnan, S. Floyd, D. Black *et al.*, "The Addition of Explicit Congestion Notification (ECN) to IP," Internet Requests for Comments, RFC 3168, September 2001.

[51] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for Data Center Networks," in *CoNEXT 2012*.

[52] 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 *SIGCOMM 2015*.

[53] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing Flows Quickly with Preemptive Scheduling," in *SIGCOMM 2012*.

[54] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal Near-optimal Datacenter Transport," in *SIGCOMM 2013*.

[55] 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 *CoNEXT 2015*.

[56] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A Centralized "Zero-queue" Datacenter Network," in *SIGCOMM 2014*.

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

[58] 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 *SIGCOMM 2015*.

[59] G. Judd, "Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter," in *NSDI 2015*.

[60] J. Bennett, C. Partridge, and N. Shectman, "Packet reordering is not pathological network behavior," *Networking, IEEE/ACM Transactions on*, vol. 7, no. 6, pp. 789–798, Dec 1999.

[61] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *IMC 2010*.

[62] S. Ghorbani, Z. Yang, B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Micro Load Balancing in Datacenters with DRILL," in *SIGCOMM 2017*.

[63] N. Yadav, M. A. Attar, S. Gandham, A. Singh, and S.-C. Chang, "Federated microburst detection," U.S. Patent, May 24, 2018.

[64] J. Sonchack, O. Michel, A. J. Aviv, E. Keller, and J. M. Smith, "Scaling hardware accelerated network monitoring to concurrent and dynamic queries with *flow," in *ATC 2018*.