A Microscopic View of Bursts, Buffer Contention, and Loss in Data Centers

Ehab Ghabashneh[‡] Yimeng Zhao[★] Cristian Lumezanu[★]

Neil Spring[★] Srikanth Sundaresan[★] Sanjay Rao[‡]

[‡]Purdue University [★]Meta

ABSTRACT

Managing data center networks with low loss requires understanding traffic dynamics at short (millisecond) time-scales, especially the burstiness of traffic, and to what extent bursts contend for switch buffer resources. Yet, monitoring traffic over such intervals is a challenge at scale.

We make two contributions. **First**, we present Millisampler, a lightweight traffic characterization tool deployed across all Meta hosts. Millisampler takes a host-centric perspective to data collection, which is scalable and allows for correlating traffic patterns with transport layer statistics. Further, simultaneous collection of Millisampler data across servers in a rack enables analysis of how synchronized traffic interacts in rack buffers. In particular, we study *contention*, which occurs when multiple bursts arrive simultaneously at the dynamically shared rack buffer.

Second, we present a data-center-scale analysis of contention, including a unique joint analysis of burstiness, contention, and loss.

Our results show (i) contention characteristics vary widely across and within a region and is influenced by service placement; (ii) contention varies significantly over short time-scales; (iii) bursts are likely to encounter *some* contention; and (iv) higher contention need not lead to more loss, and the interplay with workload and burst properties matters. We discuss implications for data center design including service placement, buffer sharing algorithms and congestion control.

1 INTRODUCTION

Modern data center networks support diverse services and communication patterns. These patterns shape network design tasks including capacity planning, fabric design, and tuning of transport parameters, all to provision an efficient network that provides low loss and latency to services. Of particular importance are the dynamics of communication at millisecond-scale intervals: traffic is typically *bursty* at these timescales, even when average link utilization is low [39].

Data center switches typically use buffers that are dynamically shared, to balance competing goals of avoiding loss—by allowing a single queue to use a significant chunk of the buffer—and of providing fairness—by preserving space for new bursts. However, little is known about the dynamics of the interaction between traffic characteristics and buffer sharing. *Contention* for the buffer occurs when multiple bursts destined to different queues arrive at the rack buffer at the same time. Because of dynamic sharing, contention results in varying amounts of buffer allocated per queue, and therefore has a significant impact on loss, latency, and ultimately, performance. Better buffer policies, and congestion control design, depend on the characteristics of traffic bursts (e.g., volume, duration) as well as the degree and variability of contention.

Existing data center traffic characterization studies focus on coarse-grained traffic characteristics such as traffic locality, the distribution of flow sizes and durations, and the utilization of links and buffers [13, 22, 25, 35, 46]. However, understanding burstiness and contention, and how they relate to losses in data center networks, requires characterizing traffic at both fine time scales and large network scales.

In this paper, we make the following **contributions.**

First, we present **Millisampler**, a lightweight network traffic characterization tool for continual monitoring that we have developed which operates at fine, and configurable time scales. Using Millisampler, we can characterize traffic across all servers within a data center rack *at the same time* (§ 4). We have validated that at the sampling rate of Millisampler, host clocks are sufficiently synchronized to ensure packets processed by the rack switch at the same time appear in simultaneous Millisampler runs collected across hosts (§4.5). We have deployed Millisampler at Meta with data collected at every host in our fleet, with the broader goal of understanding workloads, identifying difficult traffic patterns, and troubleshooting the interactions between application behavior and the network.

With Millisampler, we characterize traffic data at hosts rather than switches [46] for three reasons. First, it is easy to collect, store, and serve data at hosts. Switches are less uniformly programmable to safely collect high-frequency statistics at scale. Second, the burden on switches to collect simultaneous information on all ports is an order of magnitude or more higher. Third, instrumenting the host gives us rich context such as service information.

As a **second** contribution, we analyze Millisampler data from two data center regions of Meta over an entire day which corresponds to over 8 billion sample points.

Specifically, we present the **first large-scale characterization of data center rack buffer contention.**

We measure the degree of contention—how many bursts contend for the buffer simultaneously—across racks (§7.1) and how it

IMC '22, October 25–27, 2022, Nice, France

ACM ISBN 978-1-4503-9259-4/22/10.
https://doi.org/10.1145/3517745.3561430

varies both over a day (§7.2) and over seconds (§7.3). We find that contention characteristics vary widely across and within a region—even during busy hours in one region, 75% of racks see low average contention, while 20% see average contention higher by a factor of 3.4x due to computation-near-storage placement constraints. Further, the contention level of each rack is persistent throughout the day. Contention varies over milliseconds, often resulting in per-queue buffer reductions of 34-70% over short periods.

We also present a **joint analysis of traffic burstiness, contention and loss.** We characterize how losses in Meta data centers depend on the degree of contention (§8.1). We then characterize which bursts are most likely to encounter packet losses with regard to burst properties and the degree of contention (§8.2). Somewhat surprisingly, higher contention does not necessarily correlate with higher losses, potentially due to stable workloads and stable per-queue buffers. Nearly 92% of bursts see contention, and losses are most likely to occur with contended bursts that are a few milliseconds in duration.

Our results raise interesting questions on how service placement algorithms should incorporate network workloads, and insights on whether buffer sharing policies should vary across racks, and across time. The results also motivate the need for joint explorations into the interactions between congestion control algorithms and buffer sharing policies. Overall, the results highlight the importance of a fine grained traffic analysis approach, and the promise of Millisampler.

2 BACKGROUND AND MOTIVATION

We start by providing background on buffer contention in data centers. We then motivate why monitoring traffic patterns, especially contention and burstiness, at short time-scales is important to managing data centers with low loss. Finally, we present rationale for Millisampler, and why existing traffic measurement approaches are inadequate.

2.1 Background

2.1.1 Switch buffer. Data center networks typically use shared memory switches with a common packet buffer shared across all interfaces. To limit unfairness, the size of each queue of every interface is usually limited by a dynamic sharing mechanism that may vary across implementations. In this paper, we focus on a sharing algorithm based on the number of active queues and the amount of free space in the buffer [16] because it is deployed in all the racks that we study. Concretely, let B denote the total size of the shared buffer, and $Q(t)$ the total occupancy of the buffer at a given point in time t (i.e., the sum of the total traffic in each queue). The maximum limit $T(t)$ of each queue at time t is given by the formula below where α is a tunable parameter, which we discuss later in the section.

$$T(t) = \alpha * (B - Q(t))$$

2.1.2 Buffer contention. A data center rack typically has several servers that are actively sending and receiving traffic. Downlink from the rack, each server is mapped to a single egress queue. A queue is active when it has packets to transmit, and it uses a portion of the shared buffer which is dynamically allocated. Now consider a scenario with S active queues contending for the shared buffer, all of which exercise the buffer to their permitted limit at the same time t. Let T be the limit that each queue sees in this state. Substituting in the above equation, we have $T = \alpha * (B - ST)$, which in turn yields:

$$T = \frac{\alpha \times B}{1 + \alpha \times S}$$

Figure 1 shows how T varies for different α and S. If $\alpha = 1$ (the default value used in our fleet), then, $T = \frac{B}{1+S}$. This implies a single active queue would see a maximum queue size of $B/2$, while if two queues were active, each would see a maximum queue size of $B/3$. A higher α results in larger queue sizes for the same S – e.g., for $\alpha = 2$, the limit is $2B/3$ for a single active queue, and $2B/5$ for each of two simultaneously active queues. However, the impact of α is greatest when contention is low.

2.2 Why study contention and burstiness?

In this paper, we characterize both the *contention* in data center rack buffers and the interplay with burst properties and loss. In doing so, we are motivated by several observations:

Contention levels impact data center losses in non-trivial ways. Clearly, higher contention results in smaller per-queue buffers. Interestingly, however, the buffer available per queue is more variable at lower contention levels. This may be observed from Figure 1 where the slope (i.e., how much buffer available varies for a given variation in contention) is significantly steeper at lower contention levels across different α values. At higher contention levels, buffer share per queue is smaller, but is more stable across contention levels. Smaller buffers have a reduced ability to absorb bursts which typical congestion control algorithms that rely on RTT-timescale feedback loops also cannot control well. Loss is particularly hard on short bursty transactions because they can result in a higher fraction of loss or even retransmission timeouts. However, certain workloads may handle more stable buffers better, even if they are smaller.

Burst properties and contention jointly impact losses. While it is well known that data center traffic is bursty [46], it is less clear what properties of bursts makes them most prone to loss, and how this interacts with contention. Extremely small bursts can be absorbed by buffers, while long bursts that last several RTTs can be handled by congestion control. Thus losses seen in a data center are dictated by a combination of burst properties (e.g., lengths, volumes and number of connections), and their interplay with contention (which impacts buffer availability).

Contention and burst characteristics can guide buffer sharing policies. Understanding contention and burstiness provides important insights when tuning the dynamic buffer sharing algorithm (in particular, the parameter α) to ensure low loss. As seen from Figure 1, larger α provides a larger share to each active queue, but also results in more variability as the number of queues change. The choice of α is particularly important at lower contention levels. If contention were consistently small, and showed less variation, a larger α might be appropriate to reduce losses related to traffic bursts on the typically small number of active queues. However, significant variation in contention may argue for smaller α to ensure fairness and more stability across queues.

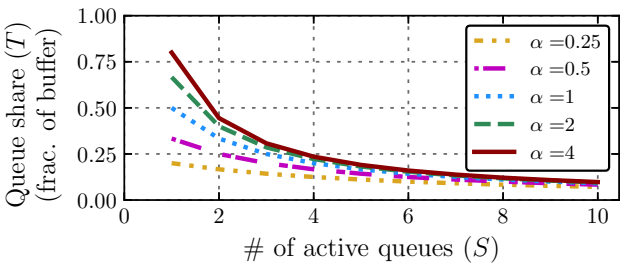


Figure 1: The maximum fraction of the buffer each queue may get for different choices of α.

2.3 Millisampler Rationale

In this paper, we study contention, and how the interplay with burstiness impacts loss on a production fleet at region scale. We achieve this through Millisampler, a tool that we have built and deployed to collect fine grained network traffic information on hosts. In designing Millisampler, we were motivated by the following observations:

Existing production switch-based traffic monitoring is coarse-grained. Conventionally, measuring data center network traffic involves coarse-grained SNMP counters or relies on sampling [13, 22, 25, 35, 46]. Unfortunately, polling counters is resource intensive, and is hence typically done at coarse time-scales (minutes). Thus, many past studies have focused on coarse-grained traffic characteristics such as locality, flow size duration and distribution, and link and buffer utilization [8, 13, 22, 25]. In contrast, bursts occur at much smaller time scales, and coarse-grained link utilization does not correlate well with loss rates [46]. It is tempting to explain high loss and low utilization in coarse-grained statistics by assuming short-duration bursts; Millisampler can confirm, and in some cases, has pointed toward other causes of loss by measuring smooth traffic.

Fine-grained switch-based traffic monitoring is resource intensive to deploy at production scale Zhang et al. [46] collect fine-grained ToR switch statistics at 10s to 100s of microseconds granularity and characterize microbursts in Facebook production data center networks. The approach involves modifying the switch platform, which involves working with proprietary and vendor-specific router ASIC SDKs. Further, heavy switch instrumentation is computationally expensive. Indeed, the data from [46] is limited to a few top-of-rack (ToR) switches collected on a one-time basis, and samples only a single port at a time. Because the work only samples a single port, it does not explore contention or how it relates to bursts and loss.

3 THE META NETWORK

Meta operates a global data center network with "data center regions" in multiple continents. Each region comprises multiple data center buildings. Each building houses a fabric-based topology, consisting of pods, which are a three-layer cluster with the top-of-rack (ToR) switch at the bottom. Each rack consists of homogeneous servers; different racks will have servers of NIC speeds typically ranging from 25 to 100 Gbps. The racks connect to upstream switches using 4 or 8 uplinks, each of 40 Gbps or 100 Gbps capacity. The ToR switches in the Meta network have dynamically shared buffers, as explained in § 2.1.

Most of the server-to-server traffic in the Meta network stays within the region. This traffic uses DCTCP as the transport protocol, while the smaller amount of inter-region traffic uses Cubic. For this work, we focus on all of the buildings in a single representative region. We focus on a single server type which uses a 50 Gbps NIC that is shared across 4 servers. Each server is allocated 12.5 Gbps, mapped to individual queues in the ToR (i.e., each server gets its own queue). Such servers represent a large plurality of Meta's network. The ToR for these servers have a buffer of size 16MB, divided into four quadrants of 4MB each. Of the 4MB, a small amount is made available as dedicated buffer for each queue, and the rest, about 3.6MB, is shared across all queues. An egress queue maps to a single quadrant as a function of the input and output port. The shared buffer has an α value of 1, which means that the maximum buffer that a single queue can consume in an otherwise empty buffer is 50%. In practice, this is a roughly 1.8MB *maximum* per queue; when there is contention, the buffer may fill sooner.

We focus on in-region (DCTCP) traffic in this work, both because it is dominant and because we can collect ECN marks to observe congestion. We note that ECN does not prevent loss; because it is still an end-to-end congestion signal, DCTCP struggles to react to short bursts that span less than a few RTTs. The problem is exacerbated in heavy incast scenarios, where even a small congestion window per sender can result in packet loss due to the large number of senders overflowing the buffer. Small bursts and heavy incast traffic patterns occur frequently in the Meta network.

The ToRs that we study have the smallest buffer in our fleet and the slowest server-link speeds; other ToR models have larger buffers and faster links (and different buffer sharing algorithms). Our focus on the smaller buffers and slowest links is because they, apart from constituting the plurality of our network fleet, also offer the best opportunity for studying pathological buffer contention. The smaller buffers lead to a higher probability of discards, and the slower draining queues lead to a greater chance of traffic contending in the buffer. The level of congestion and contention in other ToR's are comparatively less due to these reasons.

Operationally, we observe that most of the congestion in our network happens in the server-link connecting the ToR to the servers. Because of this, our ECN deployment is currently largely operational only on the ToR, resulting in DCTCP not getting ECN signals in the relatively less frequent event that congestion occurs in the higher layers of the network. We have deployed a 120 KB static ECN threshold for all our ToRs; we find that this value offers good performance across our varied workloads, though we do not claim that it is optimal. Per-service task placement is spread across racks, and not localized to single services per rack; however placement may still result in certain workloads being more predominant on some racks than others.

4 MILLISAMPLER AND SYNCMILLISAMPLER

To characterize bursts and their impact on the network, congestion control, and rack switch buffers, we measure timeseries of

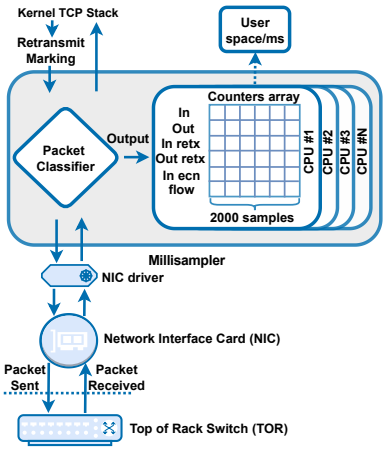


Figure 2: Millisampler architecture, with packet flow and counters.

network utilization, retransmissions, congestion-marked traffic volume, and connection counts at data center RTT granularities. To collect representative measurements at data center scale, we impose the following constraints on our measurement:

(1) Low per-packet processing, to run at line rate.
(2) Low storage overhead, to keep history of all server-links.
(3) Programmable, deployable, and maintainable for every server.

No existing approach fits all our requirements. End host packet capture tools, such as tcpdump, face performance costs in shipping packet headers from kernel to user space and potential data loss at peak traffic should the kernel-to-user buffer overrun. Conversely, switch-based monitoring is intensive on switches with limited resources, so difficult to deploy at scale, as we discussed in Section 2.3.

In this section, we describe Millisampler, a tool to collects network utilization at fine time scales with low storage and processing overhead, and SyncMillisampler that extends this to run simultaneously across all hosts in a rack.

4.1 Millisampler

Millisampler comprises user-space code to schedule runs, store data, and serve data, and an eBPF-based [1, 42] tc ("traffic classification") filter that runs in the kernel to collect fine-timescale data. The user code attaches the tc filter and enables data collection periodically. Occasional execution minimizes overhead. Because it is implemented in eBPF, it operates as compiled machine code in the kernel with direct access to kernel data. A tc filter is among the first[1] programmable steps on the receipt of a packet and near the last step on transmission. On ingress, this means that the eBPF code executes on the CPU core that is processing the soft irq (bottom half) as the packet is directed toward the owning socket. Because processing happens on many CPU cores, to avoid locks, we use per-cpu variables, which increases the memory requirement to eliminate risk of contention.

[1]One could attach kprobes to the driver or use XDP [2] programs to process inbound packets earlier; these have other issues (portability).

There are two important parameters in Millisampler: the sampling interval and the number of time buckets. The choice of sampling interval allows us to observe traffic at a wide range of granularities: we schedule runs with three values: 10ms, 1ms, and 100μs. The number of time buckets controls the memory footprint. Seeking to limit memory and storage use, we fix the number of buckets to 2000, regardless of sampling interval. This means that our observation periods range from 200ms (100μs sampling rate) to 20s (10ms sampling rate). The memory footprint of each run consists of 2000 64-bit counters per CPU core for each value we measure.

To construct timeseries, the tc filter takes as input the sampling interval and an "enabled" flag. It records as the start time of a run the timestamp of the *first* packet[2] when enabled. Millisampler determines the time bucket for the packet by determining how long it has been since the start, then dividing that interval by the sampling interval. It can then increment the relevant counters. If the computed time bucket is beyond the number of buckets, the filter will clear the enabled flag as a signal of completion to user-space and to save future processing. User code waits until the expected run time has passed and for the "enabled" flag to clear, then detaches the tc filter from the processing path and reads the counters. Detaching the tc filter ensures that no CPU time is used by the Millisampler while it is disabled. User code then stores this data in the local disk to be available on demand, and schedules another run.

4.2 Millisampler measurements

For each CPU and in each time bucket, Millisampler tallies the ingress and egress total and retransmitted bytes, ECN-marked ingress bytes, and the number of active connections.

Collecting ingress and egress total and ingress ECN-marked bytes is straightforward: the lengths and CE bits are in the packets. Existing Meta tools instrument the TCP stack and label packets as retransmissions. These tools detect when TCP processes a timeout or fast retransmission (not a tail loss probe [20]) and set an unused bit in the header of the next outgoing packet in the connection. Millisampler uses the bit to count retransmitted bytes.

Millisampler uses a 128-bit sketch [19] to estimate the number of active (incoming and outgoing) connections. This means that the connection count is an approximation that is precise up to a dozen connections and saturates at around 500 connections per sampling interval. Although there is space for additional precision, in practice, more than the actual number of connections, the qualitative variation between a few connections to dozens or hundreds of connections has been helpful toward identifying patterns of traffic with more connections (heavy incast) as opposed to more traffic on fewer connections. Because it is stateless, we lack information about whether an active flow in one interval was also active in subsequent intervals.

Multiple counters may be incremented per packet; e.g., if the ingress packet was ECN marked or retransmitted. Typically, to store all counters yields a footprint between 4–12 MB, up to 60 MB on larger machines with many cores, which is acceptably small as such larger machines tend to have more memory available. The

[2]More precisely, the filter operates not necessarily on packets, but on a socket buffer, which may be translated to or from many MTU-sized packets by the NIC's segment-reassembly features; here we use "packet" for simplicity.

storage footprint then comprises the aggregated counters from periodically executed runs, compressed and stored on the host for about a week, typically a few hundred megabytes. This week-long history permits diagnostic analysis of atypical events, including firmware bugs, kernel locking errors, and large congestion events. For instance, Millisampler helped uncover a NIC firmware bug by isolating examples of packet loss although utilization was low at fine time-scales.

4.3 Performance

We confirm that Millisampler is efficient enough to both run on every machine and to have a minimal impact on packet transmission time when active.

The in-kernel memory footprint of Millisampler is, on average, 3.6MB including counters of each type (e.g., ingress bytes), for each of 2000 samples, for each CPU core. The CPU use of Millisampler, on average, reaches 0.003%, likely because Millisampler only runs some of the time. Both in memory and CPU, Millisampler is smaller than other widely-deployed monitoring processes at Meta.

In microbenchmarks enabled by the BPF testing framework [42] on an Intel Skylake generation processor at 1.60GHz, the time to inspect and count all features of a single packet is 88 ns. The time to process does not seem to depend on how many counters are set (i.e., whether the packet has ECN) but on how many features of the packet are inspected. For example, omitting flow counting reduces the time per packet to 84ns. When the tc filter is installed (in the packet processing path) but disabled (because it is not yet enabled to run or has finished 2000 samples), the time to near-immediately return from processing is 7ns. The 88ns we observe per processed packet is substantially smaller than our best-case estimate of the time required for running tcpdump to capture the same information on the same machine. On this machine, tcpdump consumes 271ns of CPU time per packet, based on `time tcpdump -npi eth0 -s 100 -c 1000000 -w /dev/null`. In these benchmarks, the time to read the counter bpf map is a fixed 4.3ms, regardless of how many packets are counted. A fixed amount of time per run and designing for the worst, most heavily loaded case, is important for this design. With these per-packet and per-run estimates, Millisampler comes out ahead of tcpdump after just 33,000 packets. For context, we note that even in the 0.2 second duration of a 100μs sampling rate run, 33,000 packets are possible.

4.4 SyncMillisampler

Millisampler provides visibility into bursts on a single host at millisecond scale. To understand whether bursts have synchronized patterns across servers within a rack, we develop SyncMillisampler to schedule concurrent Millisampler runs. SyncMillisampler has a centralized control plane which sends data collection requests to all servers across a rack and schedules them to start collecting data at a specific time. To ensure that SyncMillisampler does not conflict with a periodic Millisampler run, we schedule SyncMillisampler data collection far enough in advance that no run will be active, then prioritize scheduled SyncMillisampler runs over periodic collection.

After all servers finish the run, the centralized control fetches and processes the compressed data from all servers. Recall that each Millisampler run will start when it observes the first packet after being enabled; collectively, each may start at a slightly different time. Each start time is recorded, so to combine these runs into a single one with uniform timestamps, we use linear interpolation to construct data points for those series that are not already aligned.

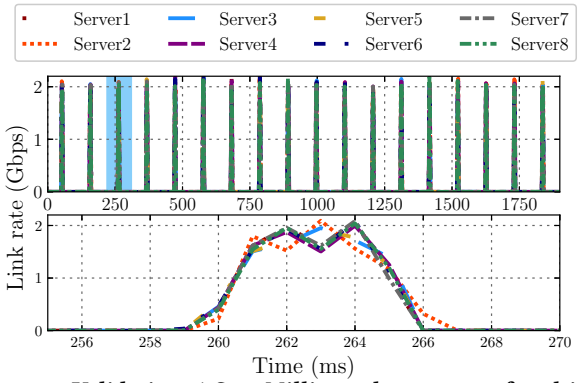


Figure 3: Validation: A SyncMillisampler capture of multicast bursts received by eight servers in one rack shows synchronization of collection across receivers.

4.5 Validation

We validate SyncMillisampler using two experiments:

Time synchronization. SyncMillisampler relies on multiple hosts starting Millisampler at the same time. Aligning these concurrent Millisampler runs requires that the host clocks be synchronized, at least to the timescale of the precision of the sampling rate of Millisampler. Such synchronization should be achievable since these hosts synchronize via one level of NTP servers to dedicated appliances with stable clocks, using interleaved NTP to achieve sub-millisecond precision [29].

To validate that packets processed by the rack switch at the same time appear in simultaneous Millisampler runs at the same time, we wrote a tool that sends periodic bursts to a rack-local multicast address. The rack switch replicates packets in the bursts, and when links are idle, these packets should arrive at subscribing hosts at the same time.

We chose a production rack where most servers are idle, subscribed eight servers to the multicast address, and run our tool to send bursts every 100ms. We collect data using SyncMillisampler at a 1ms sampling frequency since all the analysis we report in this paper are based on this sampling rate. Figure 3(top) shows the link rate per sample (1ms) for each of the subscribed servers. Figure 3(bottom) zooms into one of the synchronized periods, showing how each server received the beginning of the burst in the same sample. The lines for all servers overlap indicating the collection across hosts is synchronized. Note that the bursts do not reach the network interface line rate as multicast traffic is rate limited.

Identifying simultaneously bursty servers. We perform an additional validation experiment to show that SyncMillisampler can accurately identify the number of simultaneously bursty servers in a rack, an important component of our analysis in later sections.

The experiment is performed using a burst generator tool which we developed. The tool involves a client periodically requesting a server to transmit a burst of a specified volume. Each request is

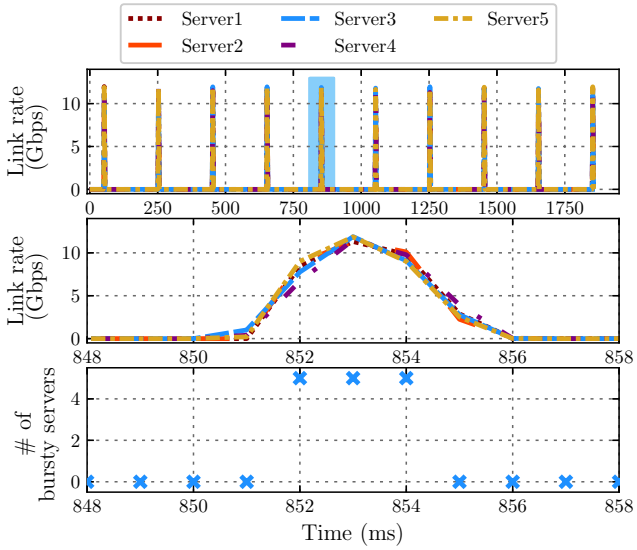


Figure 4: Validation: SyncMillisampler correctly identifies the number of concurrent bursty servers.

sent at the specified frequency based on client's local clock. We ran the tool with five clients in the same rack receiving periodic bursty traffic from five servers spread across five racks. Each burst had a volume of 1.8 MBytes, resulting in approximately 3 ms duration bursts, sufficiently long to be detected at a 1 ms granularity. Figure 4 (top) shows the link rate per sample for each client from SyncMillisampler logs when a 1ms sampling rate is used. Figure 4 (middle) zooms into one of the bursts, showing the burst volume that each client receives during the burst samples. Again, the lines overlap indicating SyncMillisampler correctly identifies that the bursts are received near simultaneously. We next ran a post-analysis on the SyncMillisampler logs to identify the number of simultaneously bursty servers. Figure 4 (bottom) shows the number of simultaneously bursty servers identified by our post analysis. The graph shows that SyncMillisampler correctly identifies that there are 5 bursty clients over the same 3 ms interval.

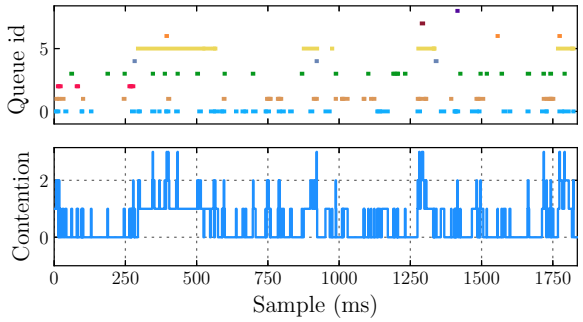
4.6 Discussion

Retransmissions observed by Millisampler indicate when losses are repaired, not when they occur. In contrast, ECN marks indicate congestion in real-time. To identify the bursts that lead to loss, our analysis must look for retransmissions that occur an RTT later.

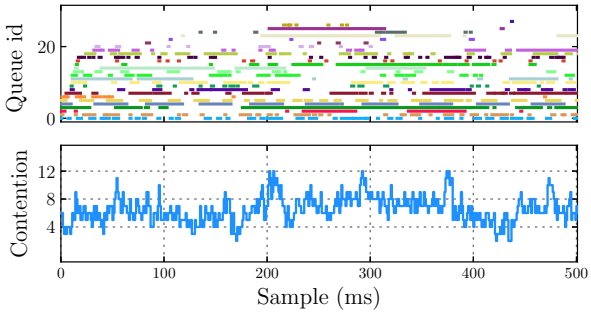
The tc layer sees segments before the sending NIC's segmentation offload and after the receiver's offloaded reassembly. Thus, the filter may see 64KB segments, potentially inflating burstiness at very fine timescales (e.g., 100μs buckets). At such rates, we often see periods of data rates in excess of line speed. We focus on collecting data with 1ms sampling intervals, avoiding this issue.

The kernel must process packets. Rarely, we have observed locking bugs in the kernel that prevent any handling of network interrupts; in these cases, Millisampler will see no data even though the network interface card is receiving, which can lead to additional apparent bursts.

The kernel must process packets. Rarely, we have observed locking bugs in the kernel that prevent any handling of network interrupts; in these cases, Millisampler will see no data even though the network interface card is receiving, which can lead to additional apparent bursts.



(a) Example run with low contention, varying between 0 (idle) and 3.



(b) Example run with high contention, varying between 3 and 12.

Figure 5: Deep dive into two runs with SyncMillisampler.

5 DATASET

In this paper, we primarily report results collected using SyncMillisampler over an entire weekday in 2022 from two different Meta regions (which we refer to as RegA and RegB), each comprising multiple data centers. We focus our presentation on RegA, and present results from RegB when the trends differ substantially. Each region consists of thousands racks of the type we study, with the average rack connected to 92 servers. We have validated our results by collecting data from each region for three additional weekdays—our results are similar, so we do not report them in the paper.

For each region, we randomly select 1000 racks to perform SyncMillisampler (§4.4) runs every hour. While Millisampler itself can sample data at different time granularities, the results in this paper use sampling rate of 1ms (over a 2s duration). This sampling rate is ideal due to several reasons. It balances duration and precision: a higher sampling rate would restrict observations to a smaller duration potentially missing information about longer bursts, while a lower sampling rate would provide less detail about short flows. Further, our switch buffers are capable of queuing about 1ms worth of packets per queue, therefore much shorter bursts are unlikely to result in loss, and therefore not interesting to us. Finally, we found in §4.5 that our clocks are sufficiently synchronized to

Region	# of runs	# of server runs	# of bursty server runs	# of bursts	# of racks
RegA	22.4K	1.98M	0.67M	19.5M	1000's
RegB	22.4K	2.1M	0.58M	23.9M	1000's

Table 1: Summary of dataset for 1 day, and we validated with 3 more days.

align at this rate. Since the collection at each server may start and end at slightly different times, we trim data to only consider the common time region. After selecting only the overlapping interval, the average length of a SyncMillisampler run is 1.85 seconds. Our primary dataset (see Table 1) includes 44.8K runs in the day, comprising 4.08M per-server runs, and in turn 8.16B sample points.

Bursts and contention. Our goal in this paper is to understand the extent to which traffic in data centers contend for shared buffer resources, and to explore the relationship between traffic patterns, buffer dynamics, and loss. Since loss is typically caused by bursty traffic patterns, we focus our analysis on traffic bursts. We define a **burst** as any consecutive set of one or more sample data points that exceeds 50% of line rate, following previous work [46]. Traffic less than this rate does not typically result in buffering. We find that 34% of server runs have bursty ingress traffic (we use ingress henceforth to refer to traffic entering a host for consistency with our earlier discussion) on server-links; 49.7% of the ingress traffic on server-links is transferred in bursts. We focus only the ingress, in-region traffic on server-links. Ingress traffic constitute the major source of packet discards in our network, and in-region traffic comprises of the vast majority of traffic. In-region traffic use DCTCP as the congestion control mechanism.

Figure 5(a) presents the time series of an example run with SyncMillisampler. The X-Axis is the 1ms sample obtained during the run, and the Y-Axis corresponds to a server queue (only servers that exhibited a burst during that run are shown). A dot indicates that a particular server was bursty in that sample. Many servers show multiple well-separated bursts, and bursts vary in length §6.

We define **contention** as the number of servers that are simultaneously bursty during each 1ms data point of the run. In Figure 5(a), vertically aligned dots correspond to simultaneously active servers. Clearly, over an entire run, the contention values obtained may vary from data point to point. Figure 5(a) shows how the contention level varies over the run. Notice that the contention level varies between 0 and 3. A contention level 0 indicates no bursts, while a contention level 1 indicates a single burst (which effectively sees no buffer contention). Recall from §2.1 that even seemingly small changes in contention level can significantly impact buffer availability to queues – for instance, in this figure, the buffer available to each server queue would vary between 0.5 and 0.25 of the total shared buffer depending on contention level. We look at different statistics of contention, such as average, or 90^{th} percentile, in our analysis as appropriate. Figure 5(b) presents an example of another sync-run where the contention levels are much higher. These graphs illustrate SyncMillisampler's ability to analyze traffic dynamics at fine granularities (§2.3).

6 A QUICK VIEW OF BURSTS

Before we delve into the effect of bursts, we present a high-level characterization of bursts in Meta's data center. Our definition of bursts excludes traffic with smooth utilization that never reaches 50%. This is deliberate: such traffic is "easy" for congestion control to handle, and unlikely to result in buffering. We focus this section on RegA, which has 19.5M bursts total. Bursts for RegB show similar trends.

Server-links are largely idle. For each bursty server run, we compute the average utilization across the run (Figure not shown). The median run average utilization is 6.4%, while the 95^{th} percentile is less than 45%. Furthermore, within bursty runs, we separate out the bursty period from the non-bursty period. Outside the bursts, the median average utilization is 5.5%. In comparison, inside bursts, the median average server-link utilization is 65.5% indicating that bursty periods only occupy a fraction of the total run.

Bursts are frequent, but short. Figure 6 shows the normalized number of bursts per second. The median run sees 7.5 bursts per second, while the 90^{th} percentile is significantly higher (39.8). Figure 7 shows the distribution of burst lengths (blue). The median burst length is 2ms, while the 90^{th} percentile length is 8ms. We will discuss the implication of bursts of this length in §8.1.

Non-contended bursts are shorter and smaller volume. We classify bursts that see contention at any point in their lifetime as **contended bursts**, and those that do not see contention at any point as **non-contended bursts**. 84.8% of bursts in RegA experience contention. Figure 7 also show the burst length for contended (red) and non-contended bursts (green). The non-contended bursts are shorter with 88% less than 3ms. We have also analyzed burst volumes and observed similar trends. Across all bursts, the median (p90) burst volume is 1.8MB (9MB). In contrast, among non-contended bursts, the median (p90) burst volume is 1MB (2.9MB). These trends are expected as longer bursts have greater volume and are more likely to encounter contention.

More connections are active inside a burst than outside. Figure 8 shows the average number of connections per sample in a run inside and outside of bursts. As expected, the number of connections during a burst is higher than the number of connections outside a burst, with a median difference of 2.7x the number of connections.

7 CHARACTERIZING CONTENTION

In this section, we characterize contention in the Meta network, motivated by the following questions:

- *What is the typical degree of contention in data center racks?* Is it similar across racks, or do we see variation?
- *How does contention within a rack vary across timescales ranging from milliseconds to an entire day?*

Typical contention levels and their short term variation help us to understand the buffer available to deal with traffic bursts and the variability of these buffers. This is in turn help us understand losses in data centers and inform the design of buffer sharing policies (§2.2). Significant differences in contention across racks indicates that buffer parameters need to be configured differently across racks, or that task placement across racks needs to be revisited. Likewise,

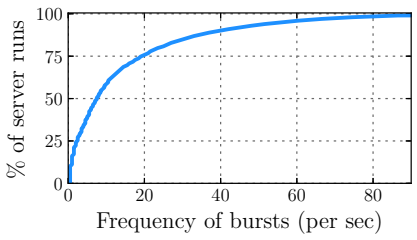


Figure 6: Frequency of bursts in a run.

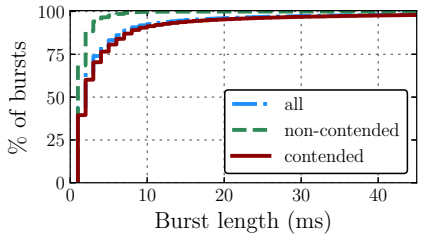


Figure 7: Bursts length distribution.

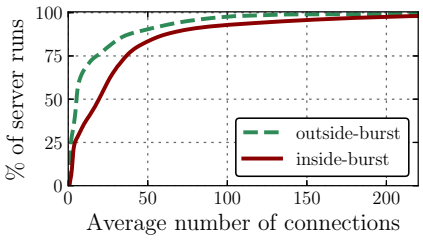


Figure 8: Connection counts.

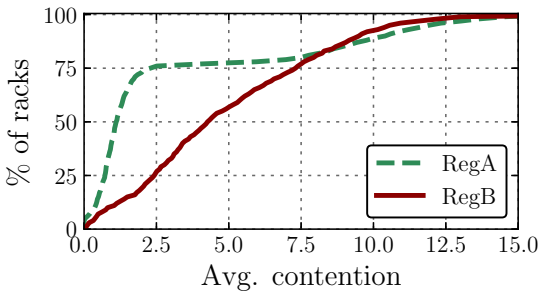


Figure 9: The average contention across racks for each of RegA and RegB during busy hour

significant variation across time (e.g., over a day) is an indication that buffer parameters may need to be modified dynamically.

7.1 How does contention vary across racks?

We start by measuring *average* contention on Meta racks. The overall load in a data center region depends on the local time, and we first present contention results for an hour that was relatively busy for both regions (6am-7am local time). We explore diurnal contention variation later. Figure 9 shows a CDF of the average contention across racks for each of RegA and RegB over this one hour period. We make two observations. First, both regions show a similar spread of average contention, however RegB shows higher contention than RegA. Second, the distribution of contention across racks of RegB is fairly uniform. Somewhat surprisingly, RegA shows a bimodal trend: 75% of the racks show relatively low average contention (less than 2.2), while the top 20% jump above 7.5. We categorize the 20% of racks in RegA with highest contention as RegA-High, and the remaining 80% as RegA-Typical (which consists primarily of low contention racks and a few with intermediate contention).

Interplay with task placement. We further investigate this unexpected bimodal trend next by better understanding the interplay with the placement of tasks on racks. In Meta, each server typically runs a single task – however, each rack may run a diverse set of tasks across the servers in the rack. Figure 10 presents a distribution of the number of tasks running on the racks in each region. The figure shows that the racks in RegA-High run significantly fewer tasks. For instance, the median rack in RegA-High only runs 8 tasks, while the median rack in RegA-Typical and RegB run 14 and 15 tasks respectively. We next consider to what extent racks are dominated by a single task. For each rack, we compute the percentage of servers in that rack which run the dominant task for that rack (i.e., the task running on the most number of servers in that rack). Figure 11 (left) presents the percentage across all the racks. The X-axis is sorted by the contention value to obtain a rack ID. The left portion of the graph corresponds to RegA-Typical racks, while the right portion corresponds to RegA-High racks. For the racks in RegA-High, the top task runs on a much larger percentage of servers (for the vast majority of these racks the percentage ranges from 60% to 100). In contrast, for most racks in RegA-Typical the top task runs on a smaller percentage of servers (the median across RegA-Typical racks is 25% , and the 90^{th} percentile is 38%). Figure 11 (right) shows RegB has a similar trend as RegA. Further analysis showed the top task in each of the RegA-High racks was the same (a machine learning task), and the results were a result of task placement that favored co-locating machine learning workloads densely in a single data center.

7.2 How does contention vary over a day?

Next, we look at how contention varies across the day, both in individual racks and across racks.

How does the contention of individual racks vary across the day? Figure 12 (top) analyzes variation across the day for RegA racks consolidating all hours (in contrast to Figure 9 which shows a single hour). Each rack is typically associated with 10 runs spread throughout the day. For each rack, we (i) determine the average contention for every run; and (ii) compute the mean, minimum and maximum of the average contention values across the runs. We sort racks by this mean to compute a rack ID, used for the x-axis. We then plot the mean (the solid line) and interval between minimum and maximum (gray interval). Figure 12 (top) shows the same qualitative bimodal distribution as Figure 9 but with 75% of racks with contention less than 1.4 and 20% with over 6.4. Further, the variation in contention for the less contended racks is small (average variation is 0.8). The highly contended racks are always highly contended; although they have higher variation (average variation is 5.3), it is not the case that they are typically low-contended with occasional outliers that draw up the average. The two categories are well separated with generally non-overlapping ranges. Overall, Figure 12 (top) shows that racks consistently experience similar average contention throughout the day. i.e., some racks persistently show significantly lower contention levels than others. Figure 12 (bottom) analyzes variation of average contention across hours for RegB. The variation is fairly pronounced, and the range of contentions for racks show far more overlap.

What diurnal trends does contention exhibit? Figure 13 shows the variation of contention by hour for racks in RegA-High (top) and RegB (bottom). We do not show RegA-Typical since the variation during the day is small, as seen earlier, and the diurnal trends mirror RegA-High.

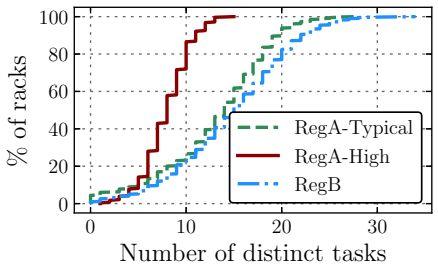


Figure 10: Tasks diversity across racks

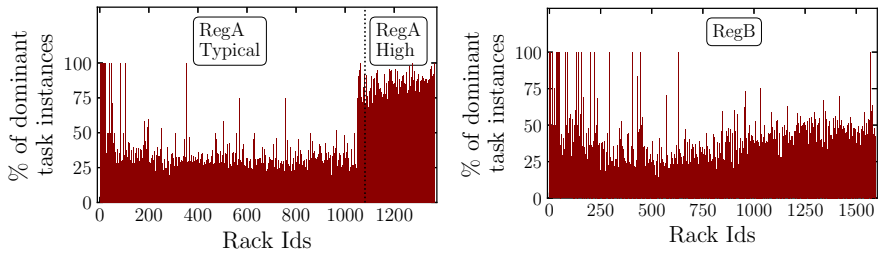


Figure 11: Dominant task density across racks.

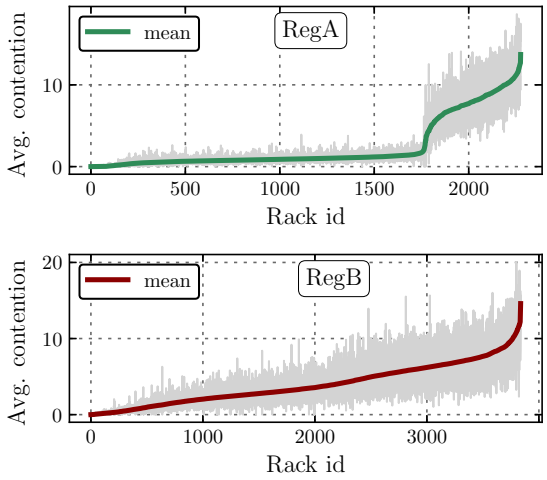


Figure 12: The variation of average contention across racks for different regions.

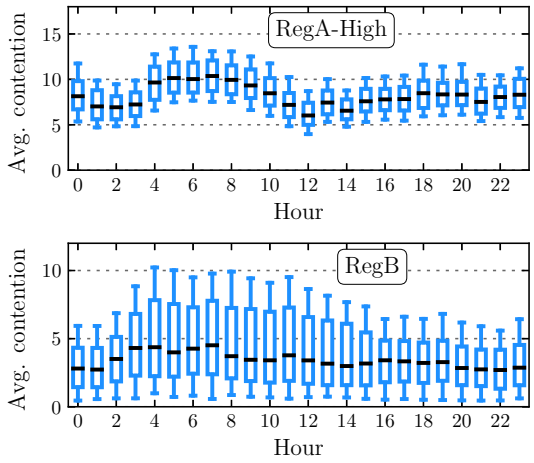


Figure 13: Diurnal trends in contention.

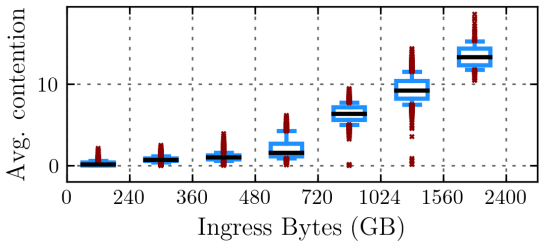


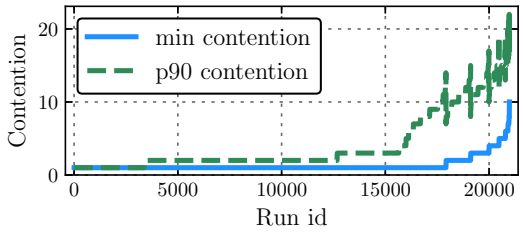
Figure 14: The correlation between average contention in a rack with total traffic entering a rack for RegA.

For each hour, we consider all runs corresponding to racks in each group, and present a box-plot that shows the variation of average contention across the runs. The diurnal pattern is strong for RegA-High: there is a clear increase in contention (27.6% on average) between hours 4 and 10. RegB also exhibits clear diurnal patterns, and the increase is particularly pronounced at higher percentiles.

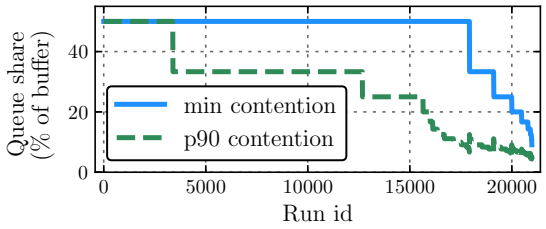
To validate the results, we also looked at diurnal trends in traffic volumes; they show similar behavior. Figure 14 examines this further for RegA by showing the correlation between average contention in a rack with its ingress traffic. The figure groups the runs into buckets based on the total ingress bytes each rack receives over 1-minute interval at which the run was collected (note that production switches at Meta only support collecting traffic volume statistics at 1 minute time granularity). The figure then plots the average contention distribution across runs in each bucket. The results show that ingress volumes do show a clear correlation with average contention. While we cannot definitively explain why the peak in traffic volumes occurred between hours 4 and 10, we note that the diurnal patterns in data center traffic depend on several factors such as background service tasks, user activity and where users are physically located, and thus may vary relative to Internet traffic.

7.3 Does contention vary over a run?

Next, we look at how contention varies within each run. This short-term variation is important as it captures the dynamics of how much buffer is available to each queue to handle a burst. We take each run and consider the minimum contention (across points with at least one active server) and the 90^{th} percentile (p90) contention. Note that we exclude 6.2% of the runs where the p90 contention is zero. Figure 15(a) shows the runs sorted by their minimum contention, followed by the p90. Figure 15(b) studies how this variation in contention impacts the per queue buffer during each run. The top curve shows the maximum buffer each queue would get during the run, while the bottom curve shows the buffer a queue would get when the p90 contention is experienced. The difference between



(a) The min and 90^{th} percentile of contention for each run from RegA.



(b) The corresponding queue buffer variation for runs on top.

Figure 15: The contention variation within RegA runs, and its impact on per queue buffer share.

the curves captures the drop in buffer share during the run. For example, runs with id between [3400-12500] experience buffer share drop from 50% to 33.3% which is a 33.4% drop from its peak. The median run encounters a drop of 33.3%, and for 15% of the runs, the drop is ≥ 70%. Note that for runs with low contention, even a small change in contention leads to significant drop in buffer share.

7.4 Takeaways

• **Contention characteristics vary widely across and within a region.** While the average contention for most racks is quite low (≤ 5), the spread of contention across racks is broad in both regions: the inter-quartile range is about 5. However, the distribution across regions is starkly different, with RegA exhibiting bimodal property with 20% of racks exhibiting high contention, owing to task placement that favored co-locating machine learning workloads in a single data center. Diurnal effects exist, but are not as dominant.
• **Contention and hence available buffer is highly variable over short time-scales.** The median run encounters a 33.3% drop in buffer share, while for 15% of the runs, the reduction exceeds 70%.

8 CONTENTION, BURSTS, AND LOSS

In the previous sections, we characterized the properties of bursts, and characterized contention in Meta. In this section, we explore how these factors together impact losses. Specifically, we are motivated by the following questions:
• *How do contention levels impact losses?* While larger contention results in smaller buffers, available buffer is more variable at lower contention levels, and may interact with congestion control in subtle ways.
• *How do burst properties and their interplay with contention impact losses?* While contention dictates the buffer available, the likelihood

Region	# of bursts	% contended	% lossy
RegA-Typical	10.2M	70.9%	1.05%
RegA-High	9.3M	100%	0.36%
RegB	23.9M	96.8%	0.78%

Table 2: Summary of the bursts in the three classes of rack.

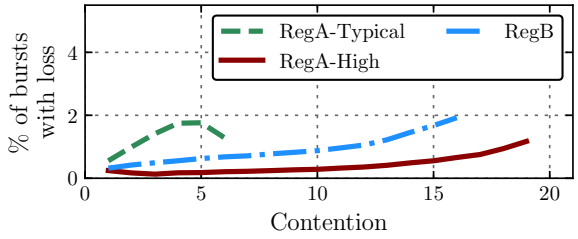


Figure 16: The correlation between contention and loss.

of loss depends on intrinsic nature of bursts (e.g., burst length, and the number of connections).

Methodology. In addition to classifying bursts based on whether they experience contention (§6), we also associate bursts with loss. Recall that retransmitted packets in our network have a bit set in the IP header (§4.2), thus making this classification straightforward. We further associate each burst with a contention level, i.e., how much contention the burst experiences during its lifetime. Specifically, we consider the contention level at each sample point of the burst, and take the maximum. We have also considered an alternate approach, where we associate each burst that experienced loss with the contention level it experienced at the time of its first loss. While bursts tend to see slightly lower contention levels at the time of their first loss compared to the maximum contention they experience, the trends are similar and we do not present these results.

8.1 How does contention level impact loss?

Table 2 presents a high level summary of the bursts in the three classes of racks; RegA-Typical, with low or moderate average contention, and RegA-High with higher average contention, both in RegA, and all racks in RegB.

First, we see that RegA-High racks are more bursty. Although this category only accounts for 20% of racks, it contributes 47.8% of all bursts. Second, across a rack, a burst is very likely to encounter contention—in fact all bursts in RegA-High, and 96.8% of bursts in RegB are classified as contended. Even in RegA-Typical, nearly 71% of bursts see contention. Third, and most surprisingly, a much larger percentage of bursts in RegA-Typical (1.05%) experience loss compared to RegA-High (0.36%), 2.9x higher. This goes against our initial hypothesis that larger contention levels should increase the likelihood of loss.

To understand this further, we associate each burst with its maximum contention level as discussed earlier. For all bursts with a given maximum contention level, we identify the fraction which experience loss, and present results in Figure 16. First, for each class of racks, the fraction of lossy bursts increases with contention levels. This is intuitive since the per queue buffer share is lower with larger contention. Second, RegA-Typical sees significantly higher loss for a given contention level. In fact, bursts are more likely to

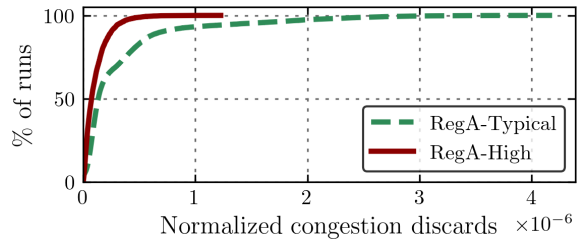


Figure 17: RegA-High racks also see fewer discards per byte in corresponding switch counters.

be lossy with RegA-Typical at contention levels under 5 relative to RegA-High at much higher contention levels.

We have also looked at congestion discards in the rack switch counters for RegA-High relative to RegA-Typical to confirm this lower loss rate despite higher contention. For each rack, we obtain the sum of per-queue congestion discards at a minute granularity, and normalize the sum to traffic volume. Figure 17 plots a CDF of the normalized congestion discards for the two classes of racks. RegA-High sees lower normalized congestion discards, consistent with Figure 16. One explanation for this result could be that since RegA-High sees more persistent contention, the end points are able to adapt better. However, RegA-High racks also correlated with congestion discards in the fabric—we believe this is due to a combination of several factors: contention occurs in the fabric due to the high density of placement of the machine learning tasks(§7.1). ASICs are more diverse, with a variety of buffer sizes, and link speeds are significantly higher (100Gbps) in the fabric. Therefore, similar contention levels could result in less loss, and also result in somewhat smoother bursts arriving downstream at the racks.

This suggests that while higher contention does lead to more loss, the relationship is more nuanced, and other factors such as workloads, location of contention, and burst properties play a role. Further, it is possible that smaller stable buffers suffice for some workloads, while other workloads may experience losses related to larger yet more variable buffers.

8.2 How do burst properties impact loss?

We now look at other burst properties (volume and number of connections) and their relation to loss and contention. We primarily focus on RegA-Typical racks: this is because it offers a good mix of contended and non-contended bursts. Second, even though contention in general is low in these racks, we saw that they suffer higher loss. We have also repeated our analysis for RegB. Although it has a much smaller percentage of non-contended bursts, the trends are similar and we do not report further.

Loss variation over bursts length. We group bursts into buckets based on their length, and find the percentage of lossy bursts per bucket. Figure 18 shows loss is initially low; this is because buffers are big enough to easily absorb them. As the burst length increases, loss increases sharply, and then stabilizes or even decreases. This is true for both contended and non-contented bursts; however, we see that beyond about a burst length of 8ms, loss is higher for contented bursts. The initial increase in loss rates for both sets of bursts can be explained because as a burst is longer,

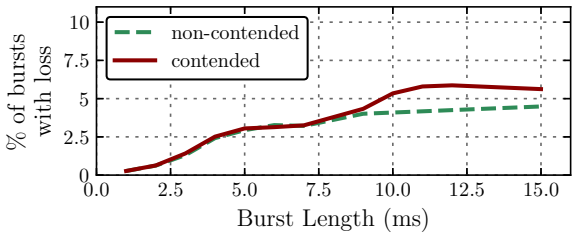


Figure 18: The correlation between burst length and loss.

it has a higher likelihood of overflowing a buffer. However, this is true only until the burst is long enough for congestion control to adapt to feedback. Contended bursts see more loss and stabilize later potentially because of higher variability in the buffers making it harder for congestion control to react before the queue overflows.

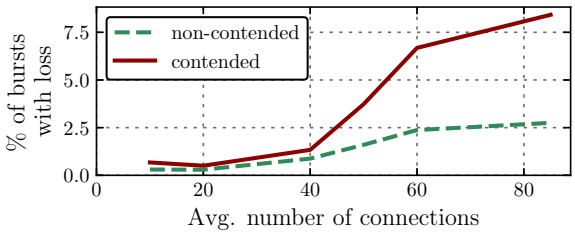


Figure 19: The correlation between incast and loss.

Loss and incast. We next look into the impact of number of connections in each burst along with contention on loss. Recall from §4.2 that connection count is based on a per-sample- point sketch and is therefore an estimate, and not the total connections for the entire burst. Figure 19 shows that the number of connections causes loss rates to increase, and then stabilize. However, we see that loss for contended bursts can be 3-4x the loss for not contended bursts. While contention considers simultaneous bursty traffic to multiple servers, and incast captures the number of simultaneous connections to the same server. Incast traffic is more likely to experience loss during contention because it will have smaller buffer available.

8.3 Takeaways

Bursts are highly likely to experience contention, and hence variable buffers. From Table 2, 91.4% of bursts experience contention, which implies lower buffer share Combined with the 8.6% of bursts that don't see contention, available buffers vary significantly.

Higher contention need not lead to more loss; burst properties matter. While loss generally increases with contention, RegA-Typical which had much less contention surprisingly experienced much more loss. We hypothesize that this stems from a combination of workloads, and burst properties such as burst length and number of connections. 4 to 6% of contended bursts of intermediate length (6-10 ms) and high connection counts (50-60) experienced loss, the regime where losses were most prevalent.

9 IMPLICATIONS FOR DATA CENTERS

Placement algorithms: Server placement algorithms in production settings do not extensively consider the impact of network

traffic patterns [33]. However, while placement can affect buffer contention (§7), and hence network performance, incorporating such patterns appears non-trivial. While the degree of contention is a potential metric to consider (which we show only loosely correlates with traffic volumes), the fact that higher contention does not translate to more loss across workloads indicates the need for more detailed metrics that combine burst properties and contention.

Buffer sharing algorithms: Since placement algorithms cannot guarantee homogeneous characteristics, our results suggest there is a strong case to tailoring buffer sharing policies to groups of racks, and to distinct workloads. A relatively small set of configurations—say one each for low contention and high contention regimes, and for certain large workloads—appear sufficient. While our results do show diurnal effects, it is less clear whether changing policies for a rack over time-scales of a day is worthwhile (as the same racks continue to have similar contention trends). However, this may need further investigation with more data.

Interplay of buffer sharing algorithms with congestion control: The variation of available buffer over RTT timescales argues for congestion control mechanisms that can explicitly handle variability in buffer, and the need for models to understand the trade-offs between smaller stable buffers, and larger but more variable buffers. The results also call for new buffer sharing algorithms designed to reduce this variability at low contention levels. Implications for newer ASICs that support larger buffers [45] but may need to deal with larger traffic volumes is another interesting direction.

10 RELATED WORK

We discussed data center traffic measurements work in §2.3 [13, 22, 25, 35, 46]. We discuss other related work:

Buffer sharing algorithms. Our study was motivated by recent advances in buffer sharing algorithms. Shan *et al.* [38] modified the dynamic threshold algorithm (DT) [16] by relaxing the fairness constraint allowing microbursts to fully use the buffer in order to avoid loss. Apostolaki *et al.* [7] generalize DT and use different α for the same queue based on flow class (i.e. higher α for mice, and lower α for elephant flows). and subsequently [6] design a new algorithm that provides better isolation for high priority traffic while sustaining link utilization. Huang *et al.* [23] dynamically assign α (per port) to control the fraction of buffer that each port can get. Other works [6, 7, 23, 38] seek to absorb bursts contending for the buffer and avoid loss. However, all proposed algorithms are designed oblivious to how contention looks inside a data center, and how contention characteristics correlate with loss. Our work can inform the design of such buffer sharing algorithms. Several works have explored buffer sizing in wide area networks [9, 12, 18, 30] but do not consider buffer contention, an important focus of this paper.

Burst characterization and microburst detection. Traditional sampling-based techniques [17, 34] are too coarse-grained to detect microbursts. Several works [14, 24, 32, 40, 41] leverage advances in programmable dataplanes to detect and measure microbursts in switches. In contrast, we focus on understanding how burst properties and contention impact loss. While we focus on data center traffic, much work (notably [27]) has analyzed the burstiness of Internet traffic.

Data center congestion control and loss diagnosis. There has been much work on data center congestion control [3–5, 15, 21, 26, 31, 37, 44, 47]. Recent work [28] has explored more responsive congestion control at short timescales, important given bursts of intermediate length are particularly challenging to tackle. Wei *et al.* [11] explore the interplay between buffer sharing and congestion control, an area that may need more attention. Others [10, 36, 43] combine end host instrumentation with probing to identify problematic links in a data center that result in loss. For instance, [10] detects TCP retransmissions at end hosts which triggers active probes to aid diagnosis. In contrast, SyncMillisampler does not create additional probes, and allows us to understand how workloads across servers in a rack lead to contention losses in router buffers.

11 CONCLUSION

We have made two contributions. First, we have presented Millisampler, a lightweight network traffic characterization tool, that takes a unique host-centric perspective to data collection. Millisampler allows for analyzing traffic at the time-scales of milliseconds at data center scale and has been operationally deployed across all hosts in the Meta network. Second, we present a data-center-scale analysis of contention in racks, including a unique joint analysis of contention, traffic burstiness, and loss. Our results highlight the importance of characterizing traffic dynamics at short time-scales, the effectiveness of Millisampler in doing so, and sheds important insights that can guide the design of buffer sharing, congestion control, and server placement algorithms.

12 ACKNOWLEDGEMENTS

We thank our shepherd, Phillipa Gill, and the anonymous reviewers for their feedback which greatly helped improve the paper. We also thank colleagues whose feedback greatly improved the work, including Rob Sherwood, Jakub Kicinski, Balasubramanian Madhavan, Chris Canel, Prashanth Kannan, Nicolaas Viljoen, and Raghu Nallamothu. This work was funded in part by the National Science Foundation (NSF) Award 1910234.

REFERENCES

[1] [n. d.]. tc-bpf(8) manual page. ([n. d.]). https://man7.org/linux/man-pages/man8/tc-bpf.8.html.
[2] Paolo Abeni. 2018. Achieving high-performance, low-latency networking with XDP. (Dec. 2018). https://developers.redhat.com/blog/2018/12/06/achieving-high-performance-low-latency-networking-with-xdp-part-1/.
[3] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM Conference*. 63–74.
[4] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. 2012. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 253–266.
[5] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 435–446.
[6] Maria Apostolaki, Vamsi Addanki, Manya Ghobadi, and Laurent Vanbever. 2021. FB: A Flexible Buffer Management Scheme for Data Center Switches. *CoRR* abs/2105.10553 (2021). arXiv:2105.10553 https://arxiv.org/abs/2105.10553
[7] Maria Apostolaki, Laurent Vanbever, and Manya Ghobadi. 2019. FAB: Toward Flow-Aware Buffer Sharing on Programmable Switches. In *Proceedings of the 2019 Workshop on Buffer Sizing (BS '19)*. Association for Computing Machinery, New York, NY, USA, Article 2, 6 pages. https://doi.org/10.1145/3375235.3375237
[8] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. 2004. Sizing Router Buffers. In *Proceedings of the ACM SIGCOMM Conference*. 281–292. https://doi.org/10.1145/1015467.1015499

[9] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. 2004. Sizing Router Buffers. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '04)*. Association for Computing Machinery, New York, NY, USA, 281–292. https://doi.org/10.1145/1015467.1015499

[10] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang (Harry) Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 2018. 007: Democratically Finding the Cause of Packet Drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 419–435. https://www.usenix.org/conference/nsdi18/presentation/arzani

[11] Wei Bai, Shuihai Hu, Kai Chen, Kun Tan, and Yongqiang Xiong. 2021. One More Config is Enough: Saving (DC)TCP for High-Speed Extremely Shallow-Buffered Datacenters. *IEEE/ACM Transactions on Networking* 29, 2 (2021), 489–502. https://doi.org/10.1109/TNET.2020.3032999

[12] Neda Beheshti, Yashar Ganjali, Monia Ghobadi, Nick McKeown, and Geoff Salmon. 2008. Experimental Study of Router Buffer Sizing. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement (IMC '08)*. Association for Computing Machinery, New York, NY, USA, 197–210. https://doi.org/10.1145/1452520.1452545

[13] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement*. Association for Computing Machinery, 267–280.

[14] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, and Ori Rottenstreich. 2018. Catching the microburst culprits with snappy. In *Proceedings of the Afternoon Workshop on Self-Driving Networks*. 22–28.

[15] Inho Cho, Keon Jang, and Dongsu Han. 2017. Credit-Scheduled Delay-Bounded Congestion Control for Datacenters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 239–252. https://doi.org/10.1145/3098822.3098840

[16] A.K. Choudhury and E.L. Hahne. 1998. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Transactions on Networking* 6, 2 (1998), 130–140. https://doi.org/10.1109/90.664262

[17] Benoit Claise, Ganesh Sadasivan, Vamsi Valluri, and Martin Djernaes. 2004. Cisco systems netflow services export version 9. (2004).

[18] A. Dhamdhere, H. Jiang, and C. Dovrolis. 2005. Buffer sizing for congested Internet links. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, Vol. 2. 1072–1083 vol. 2. https://doi.org/10.1109/INFCOM.2005.1498335

[19] Cristian Estan, George Varghese, and Mike Fisk. 2003. Bitmap Algorithms for Counting Active Flows on High Speed Links. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement*. 153–166. https://doi.org/10.1145/948205.948225

[20] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. 2013. Reducing Web Latency: The Virtue of Gentle Aggression. In *Proceedings of the ACM SIGCOMM Conference*. 159–170. https://doi.org/10.1145/2486001.2486014

[21] Keqiang He, Eric Rozner, Kanak Agarwal, Yu (Jason) Gu, Wes Felter, John Carter, and Aditya Akella. 2016. AC/DC TCP: Virtual Congestion Control Enforcement for Datacenter Networks. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 244–257. https://doi.org/10.1145/2934872.2934903

[22] Yihua He, Nitin Batta, and Igor Gashinsky. 2019. Understanding switch buffer utilization in CLOS data center fabric. (2019). http://buffer-workshop.stanford.edu/papers/paper25.pdf

[23] Sijiang Huang, Mowei Wang, and Yong Cui. 2021. Traffic-aware Buffer Management in Shared Memory Switches. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488849

[24] Raj Joshi, Ting Qu, Mun Choon Chan, Ben Leong, and Boon Thau Loo. 2018. BurstRadar: Practical real-time microburst monitoring for datacenter networks. In *Proceedings of the 9th Asia-Pacific Workshop on Systems*. 1–8.

[25] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. 2009. The nature of data center traffic: measurements & analysis. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement*. Association for Computing Machinery, 202–208.

[26] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 514–528. https://doi.org/10.1145/3387514.3406591

[27] W. Leland and D. Wilson. 1991. High time-resolution measurement and analysis of LAN traffic: Implications for LAN interconnection. In *IEEE INFCOM '91*. IEEE Computer Society, 1360–1366. https://doi.org/10.1109/INFCOM.1991.147663

[28] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: High Precision Congestion Control. In *Proceedings of the ACM SIGCOMM Conference*. 44–58. https://doi.org/10.1145/3341302.3342085

[29] Miroslav Lichvar and Aanchal Malhotra. 2021. NTP Interleaved Modes. IETF Internet Draft: draft-mlichvar-ntp-interleaved-modes-07. (oct 2021). https://datatracker.ietf.org/doc/html/draft-ietf-ntp-interleaved-modes-07.

[30] Nick McKeown, Guido Appenzeller, and Isaac Keslassy. 2019. Sizing Router Buffers (Redux). *SIGCOMM Comput. Commun. Rev.* 49, 5 (nov 2019), 69–74. https://doi.org/10.1145/3371934.3371957

[31] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-Based Congestion Control for the Datacenter. *SIGCOMM Comput. Commun. Rev.* 45, 4 (aug 2015), 537–550. https://doi.org/10.1145/2829988.2787510

[32] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-directed hardware design for network performance monitoring. In *Proceedings of the ACM SIGCOMM Conference*. 85–98.

[33] Andrew Newell, Dimitrios Skarlatos, Jingyuan Fan, Pavan Kumar, Maxim Khutornenko, Mayank Pundir, Yirui Zhang, Mingjun Zhang, Yuanlai Liu, Linh Le, Brendon Daugherty, Apurva Samudra, Prashasti Baid, James Kneeland, Igor Kabiljo, Dmitry Shchukin, Andre Rodrigues, Scott Michelson, Ben Christensen, Kaushik Veeraraghavan, and Chunqiang Tang. 2021. RAS: Continuously Optimized Region-Wide Datacenter Resource Allocation. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 505–520. https://doi.org/10.1145/3477132.3483578

[34] Peter Phaal, Sonia Panchen, and Neil McKee. 2001. RFC3176: InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. (2001).

[35] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the social network's (datacenter) network. In *Proceedings of the ACM SIGCOMM Conference*. Association for Computing Machinery, 123–137.

[36] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, and Alex C. Snoeren. 2017. Passive Realtime Datacenter Fault Detection and Localization. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 595–612. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/roy

[37] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, and Amin Vahdat. 2020. Annulus: A Dual Congestion Control Loop for Datacenter and WAN Traffic Aggregates. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 735–749. https://doi.org/10.1145/3387514.3405899

[38] Danfeng Shan, Wanchun Jiang, and Fengyuan Ren. 2015. Absorbing microburst traffic by enhancing dynamic threshold policy of data center switches. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. 118–126. https://doi.org/10.1109/INFOCOM.2015.7218374

[39] Danfeng Shan and Fengyuan Ren. 2017. Improving ECN marking scheme with micro-burst traffic in data center networks. In *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 1–9.

[40] D. Shan, F. Ren, P. Cheng, R. Shu, and C. Guo. 2018. Micro-Burst in Data Centers: Observations, Analysis, and Mitigations. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. 88–98. https://doi.org/10.1109/ICNP.2018.00019

[41] John Sonchack, Oliver Michel, Adam J Aviv, Eric Keller, and Jonathan M Smith. 2018. Scaling hardware accelerated network monitoring to concurrent and dynamic queries with *flow. In *2018 USENIX Annual Technical Conference*. 823–835.

[42] Alexei Starovoitov. 2017. bpf: program testing framework. (March 2017). https://lwn.net/Articles/718784/.

[43] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. 2016. Simplifying Datacenter Network Debugging with PathDump. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 233–248. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/tammana

[44] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. 2012. Deadline-aware datacenter TCP (d2tcp). *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 115–126.

[45] Jim Warner. [n. d.]. packet buffers. ([n. d.]). https://people.ucsc.edu/~warner/buffer.html.

[46] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-resolution measurement of data center microbursts. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference*. 78–85.

[47] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 523–536. https://doi.org/10.1145/2785956.2787484