Understanding Incast Bursts in Modern Datacenters

Christopher Canel
ccanel@cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Balasubramanian Madhavan
bmadhavan@meta.com
Meta
Menlo Park, CA, USA

Srikanth Sundaresan
ssundaresan@meta.com
Meta
Menlo Park, CA, USA

Neil Spring
ntspring@meta.com
Meta
Menlo Park, CA, USA

Prashanth Kannan
pkannan@meta.com
Meta
Menlo Park, CA, USA

Ying Zhang
zhangying@meta.com
Meta
Menlo Park, CA, USA

Kevin Lin
kevinli3@alumni.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Srinivasan Seshan
srini@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract

In datacenters, common incast traffic patterns are challenging because they violate the basic premise of bandwidth stability on which TCP congestion control convergence is built, overwhelming shallow switch buffers and causing packet losses and high latency. To understand why these challenges remain despite decades of research on datacenter congestion control, we conduct an in-depth investigation into high-degree incasts both in production workloads at Meta and in simulation. In addition to characterizing the bursty nature of these incasts and their impacts on the network, our findings demonstrate the shortcomings of widely deployed window-based congestion control techniques used to address incast problems. Furthermore, we find that hosts associated with a specific application or service exhibit similar and predictable incast traffic properties across hours, pointing the way toward solutions that predict and prevent incast bursts, instead of reacting to them.

CCS Concepts

• **Networks** → **Network measurement**; **Data center networks**; **Transport protocols**.

Keywords

Incast; microbursts; TCP; congestion control; datacenter networks

ACM Reference Format:
Christopher Canel, Balasubramanian Madhavan, Srikanth Sundaresan, Neil Spring, Prashanth Kannan, Ying Zhang, Kevin Lin, and Srinivasan Seshan. 2024. Understanding Incast Bursts in Modern Datacenters. In *Proceedings of the 2024 ACM Internet Measurement Conference (IMC '24), November 4–6, 2024, Madrid, Spain.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3646547.3689028

IMC '24, November 4–6, 2024, Madrid, Spain
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0592-2/24/11
https://doi.org/10.1145/3646547.3689028

1 Introduction

Modern datacenter networks face many-to-one traffic patterns known as *incast* [2, 6, 23]. Applications (i.e., *services*) are often organized into a partition/aggregate traffic pattern where a coordinator server dispatches up to thousands of sub-tasks to worker servers and waits for their replies [2, 18, 26]. In an effort to improve machine utilization, service architects determine fan-in degree based on host CPU processing capabilities, often creating situations where hundreds or thousands of workers interact with a single coordinator. The roughly synchronized responses from the many workers cause congestion in the coordinator's top-of-rack (ToR) switch, which absorbs excess packets in queues, increasing end-to-end latency and delaying congestion feedback to senders. Packet loss occurs when switch queues eventually overflow, causing high tail latency that directly impacts service-level performance. At endhosts, these delays and losses interact poorly with the Transmission Control Protocol's (TCP) [12] congestion control and loss recovery algorithms, resulting in poor end-to-end performance.

We first present a measurement study that makes two contributions: 1) while incast is well studied, we revisit this behavior for modern workloads in a large datacenter operated by Meta, a major Internet content provider; 2) our observations reinforce conventional wisdom but also uncover trends and characteristics of more recent incast patterns that impact the performance of current solutions. We show that incast often occurs on the scale of hundreds of flows — an order of magnitude greater flow count than most TCP-based protocol designs [2, 6, 21, 30, 32] have focused on — and is *bursty*, dissipating in milliseconds, yet still induces significant queuing, packet delay, and retransmissions. This reinforces recent work [15] showing that traffic patterns can experience large-scale incast bursts at millisecond timescales. Our study reveals that for each service, the flow count distribution in an incast is stable, and therefore predictable, both over time and across the hosts in the service. Therefore, rather than *reacting* to incast bursts as in TCP congestion control, hosts could *predict* the scale of congestion and adjust their rates proactively.

Second, we show using detailed simulations *why* congestion control breaks down under such situations; specifically, we study Data

Center TCP (DCTCP) [2], a congestion control algorithm (CCA) that is widely deployed in datacenters. We show that DCTCP cannot converge to a sufficiently small congestion window (CWND, the volume of data in flight) as the degree of incast increases, resulting in persistently high buffer utilization. We find that the combination of high flow count and the cyclic nature of bursts invites oscillations and unfairness across flows that prevent protocols like DCTCP from converging stably. Instead, divergence at burst boundaries exacerbates the next burst. These measurements imply that sender CCAs are ill-equipped to address incast on their own.

This paper proceeds as follows. Section 2 describes the datacenter environment. Section 3 presents a measurement study of production workloads, finding that incast bursts routinely experience hundreds of connections and overwhelm the reactivity of sender-based DCTCP. Section 4 uses simulations to diagnose DCTCP's shortcomings, finding that divergence at burst boundaries exacerbates the impacts of incast. Section 5 concludes with reflections on existing techniques and design considerations for future techniques.

2 Measurement environment

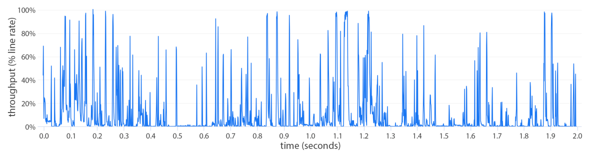
In order to better understand our measurements and observations, we provide some high-level details about our datacenter environment. The Meta datacenter network is described in [15]. Machines are organized into a three-layer topology. Each rack consists of a set of servers connected to a ToR switch using Ethernet links of 25–100 Gbps. The ToR switches in our network employ dynamically shared buffers and support active queue management techniques such as Explicit Congestion Notification (ECN) [13] marking.

The servers we measure use DCTCP [2], a network-assisted CCA that has been studied extensively. DCTCP leverages ECN [13], which enables switches to inform endpoints of congestion by setting the Congestion Experienced (CE) bits in the IP header. The receiver reflects these marks back to the sender by setting the ECE bit in the TCP header of acknowledgment (ACK) packets. DCTCP senders adjust their rate proportionally to the fraction of ACKs that are marked. We consider DCTCP for three reasons. First, its focus on keeping queues short reduces drops and improves sharing in shallow-buffered switches. Second, among datacenter CCAs, DCTCP stands out as a mature design that has been implemented in the Linux kernel since 2014. Third, the required switch features for ECN marking are well supported by switch vendors. Recent work describes our DCTCP deployment in more detail [9].

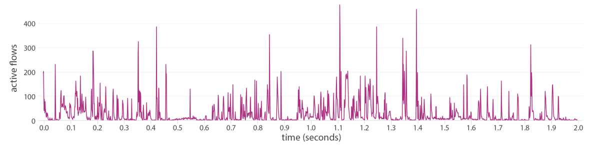
DCTCP uses two tunable parameters, the ECN marking threshold in switches, which controls the amount of queuing allowed before a switch is considered congested, and the g parameter on senders, which controls how quickly its congestion model evolves. For the ECN threshold, we use a higher value than the DCTCP paper [2] recommends to avoid underutilization when faced with host burstiness. While we avoid reporting a specific value, the chosen ECN threshold is 6.7% of queue capacity. We set the g parameter to 1/16 based on Equation 15 in [2].

3 Measuring incast bursts

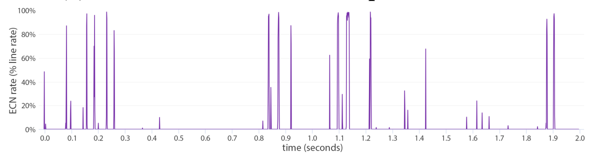
To characterize the behavior of incast bursts in modern workloads, we first show an illustrative example, then present a measurement study of five services at scale to make observations about loss rate, incast frequency and duration, queue length, and ECN marking.



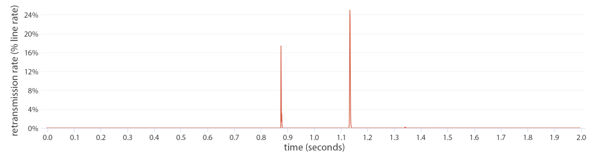
(a) Throughput is dominated by 1–5 ms bursts.



(b) The number of active flows spikes to 100–400.



(c) Packets are marked with ECN, showing queue buildup.



(d) Retransmissions are significant, but rare.

Figure 1: Example of incast bursts, measured at a receiver. Figures show two seconds, measured every 1 ms.

3.1 Utilization and loss rate

We define a "burst" as any contiguous time span where the average aggregate ingress data rate, measured at the *receiver* at 1 ms intervals, is greater than 50% of the network interface card (NIC) line rate. Figure 1 shows a sample two-second traffic trace collected from a popular service ("aggregator" below). Figure 1a shows the ingress throughput at the receiver. We see that the traffic has many sharp bursts at line rate, with each burst lasting only a few milliseconds. Average link utilization is low, only 10.6%, with essentially all traffic being part of a burst. This low overall utilization is consistent with recent measurement work [15]. Bursts are associated with high active flow counts (Figure 1b), which jump to 200 or more, signifying that these are incasts. The observation that most traffic is incast bursts motivates why congestion control behavior during incast bursts is important for overall network performance.

Figure 1c measures the ingress rate consisting of ECN-marked packets. Recall from Section 2 that switches mark packets with ECN when queues are 6.7% full. Here, we see that if traffic is marked, then essentially all packets are marked (marking rate roughly equals line rate). This means that during bursts, sender-side DCTCP will observe either no marking or high marking, leading to oscillations between ramping quickly up and ramping quickly down. Figure 1d shows retransmitted packets; those which are dropped by a full

Table 1: Five example services.

Service	Description
Storage	Distributed key-value store
Aggregator	Collects content to display on a page
Indexer	Indexing service for recommendations
Messaging	Distributed real-time messaging system
Video	Video analytics service

ToR queue. Not all bursts cause retransmissions, but when it does occur, loss can be catastrophic and reach 24% of line rate. While isolated, these loses are concerning because they impact application tail latency and therefore user experience.

3.2 Burst frequency and duration

We show that the traffic patterns described above are common across many services. We observe that compared to existing work, these workloads experience incast on the scale of hundreds, not tens, of flows. We instrument five popular services with different characteristics, described in Table 1 and chosen because of their high retransmissions. We collect a two-second trace (measured at 1 ms granularity) from 20 hosts in each service, nine times throughout a day. We collect this data using Millisampler [15, 29], a lightweight host-side measurement tool that runs as an eBPF tc filter [11, 27]. We then identify all bursts, as defined above. We consider the behavior during bursts (ignoring inter-burst periods, where utilization is low), so in the following cumulative distribution functions (CDFs), each sample corresponds to one burst.

Figure 2 measures the extent of incast bursts. We look first at burst frequency in Figure 2a: depending on the service, the receiver experiences between tens and as many as 200 bursts per second. Each burst lasts 1–20 ms, as shown in Figure 2b, with about 60% of bursts being either 1 or 2 ms — note that these measurements are at 1 ms granularity, so we cannot detect bursts shorter than 1 ms. Burst duration is in line with recent work measuring bursts in datacenters [15], and is shorter on average than those discussed in the DCTCP paper, which considers queries that complete in 2–40 ms. Importantly, given that a burst of several milliseconds corresponds to tens or hundreds of datacenter round trips, each DCTCP sender should be able to converge to an appropriately low rate. We discuss later why this is not the case. Burst duration informs the type of techniques that may be used to address it: for example, while Swift [19] supports thousand-flow incasts, these short bursts would be challenging for its pacing mode (Section 5.2). The bursts observed here are distinct from the *microsecond*-scale incasts described in [33], which are often absorbed by ToR buffers and often complete faster than congestion control can react to. An earlier study of the Meta (Facebook) datacenter network [28] noted millisecond-scale burstiness, but this study goes further to directly quantify the duration, frequency, and impacts of this burstiness.

3.3 Incast degree distribution is stable

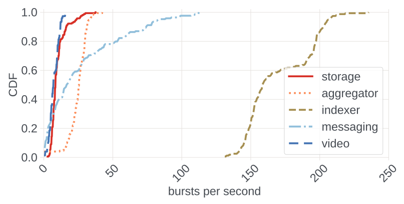
Figure 2c shows the number of *active* flows during each burst. Historically, "incast" has referred to any situation where multiple flows converge, however because using multiple flows is standard practice on high-bandwidth datacenter networks and modern CCAs perform well with a few dozen flows, we choose to define an incast with a higher threshold of 25 flows. The majority of bursts in these distributions are incasts (more than 25 flows), with some reaching as high as 200–500 flows in the 99^{th} percentile. The number of active flows is measured over a 1 ms interval, so across the entire multi-millisecond burst, more flows may have been active, just at non-overlapping times. Figure 2c also shows cases where a burst is not an incast: "storage" and "aggregator" both show a cliff where between 10% and 45% of bursts experience less than 20 flows. These bursts likely correspond to a bimodal workload where the service performs both a high-flow task like aggregating responses and a low-flow task like checkpointing.

The majority of these incasts consist of many more flows than the O(50)-flow scenarios considered by most existing approaches, such as DCTCP [2], ICTCP [32], D^2TCP [30], On-Ramp [21], and [6]. Swift [19] and recent receiver [7, 14, 16, 22] and INT-based [1, 3, 20, 31] techniques do consider hundreds or thousands of flows, but are challenging to deploy due to their requirements for fine-grained timestamping, endpoint stack modifications, or switch features. Other work measuring datacenter bursts either focuses on burst duration but not flow count [4] or considers at most 10 flows[17]. The earlier Meta (Facebook) study [28] revealed that *senders* communicate with up to 300 destination racks (*out*cast), but we investigate the number of flows experienced by each *receiver* (*in*cast). The earlier study remarked that measurement overheads prevented the evaluation of incast or microbursts. These challenges were overcome for this work (due in part to the tools described in [15]).

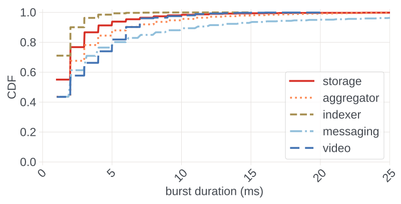
Since flow count defines the scale of an incast, we conduct a more extensive characterization. For each service, we measure active flows on 20 hosts for two seconds at 10 minute intervals over 18 hours. Earlier, Figure 2c showed a distribution of flow counts for each service. Figure 3 examines how these distributions change over time and across hosts. We see in Figure 3a that over time, the *average* flow count for each service oscillates around steady (albeit different) operating points. An exception is "video", which shifts between two operating modes of ≈225 and ≈275 flows, likely due to the service scheduler spooling up more worker hosts in response to load. Figure 3b shows that for "aggregator", selected because it causes particularly high queuing (Figure 4a), the average and p99 flow count are stable across hosts as well. While averages hide the variance in Figure 2c, the stability in Figure 3 suggests that the distributions as a whole change little on the timescale of hours. Stability in the p99 is particularly useful because it indicates the worst-case incast that a service can expect, and these worst-case bursts can have a significant impact on other traffic due to induced queuing delay and drops. Incast solutions can leverage this stability as predictability into how to prepare for future incasts.

3.4 Queuing and ECN marking

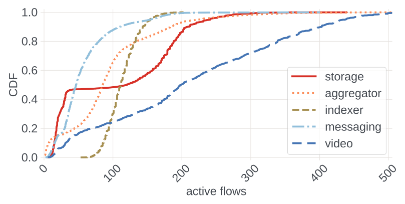
Figure 4 shows the impact of incast bursts on the network. Queue length is a proxy for increased network latency, which applications (and users) often perceive as high response time. The median burst occupies a peak of 20–100% of ToR queue capacity, as shown in Figure 4a. To reduce measurement overheads, switches record queue utilization as a "high watermark" over the last minute, which is



(a) Frequency: 10s–100s of bursts/second.

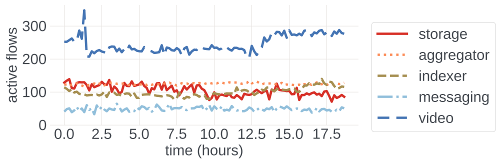


(b) Duration: Bursts last 1–20 ms.

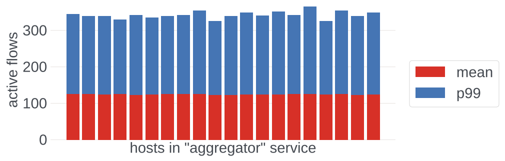


(c) Flows: Incast of up to 500 flows.

Figure 2: Incast burst characteristics across five production services.



(a) The average flow count is stable over hours.



(b) 20 sampled hosts in "aggregator" see similar average and p99 flow counts; other services are also stable.

Figure 3: Within a service, the distribution of flow count during a burst is stable over time and across hosts.

the highest occupancy the queue achieved over that minute. ToRs enforce a per-queue capacity limit, but the capacity available at runtime may be lower because total memory is shared between ports. The DCTCP paper [2] discusses how flows traversing other ports reduce buffer capacity available to absorb bursts.

Keeping the ECN marking rate low is crucial for controlling network latency and is also a measure of CCA effectiveness: DCTCP strives for the queue length to oscillate around the marking threshold, so a high marking percentage indicates that DCTCP struggles to converge. Figure 4b shows that at the 95^{th} percentile of bursts, between 2.5% and 80% of traffic is marked with ECN. "Aggregator" and "video" experience a p90 marking rate above 60%, suggesting that DCTCP struggles to keep queues short for some bursts. Note that these two services show similar marking rate distributions despite their flow counts differing greatly (Figure 2c). In this figure the y-axis starts at the 50^{th} percentile, showing that approximately 50% of bursts do not experience any marking at all, which matches the example in Figure 1c where only some bursts cause queues to build. As described in [15], simultaneous burst events to other hosts on the same rack (i.e., rack-level contention) can consume shared switch memory and likely exacerbates a subset of incast bursts.

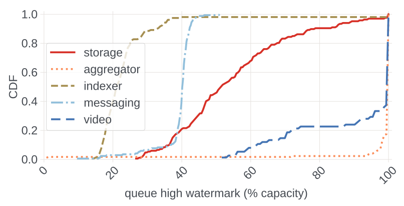
3.5 Retransmissions

In general, retransmissions occur in only a small fraction of bursts, but when they do occur, they can be significant. Figure 4c shows the volume of retransmitted data as a percentage of the NIC line rate. For the profiled services, only 5% of bursts see retransmissions (the y-axis starts at the 95^{th} percentile). The CDF is dominated by its tail: in the top 0.1% of bursts, drops result in up to 8% of data volume being retransmitted traffic. The bandwidth consumed by these retransmissions would be of better use in delivering new data. For services, this long tail of retransmissions is challenging because it leads to high tail latency for queries, although we do not present service-level latency measurements here. These results match with the observations in Figure 1d, where most bursts do not cause retransmissions, but when they do occur, they are up to 24% of line rate. These high retransmissions indicate that the CCA struggles to control congestion during incast bursts.

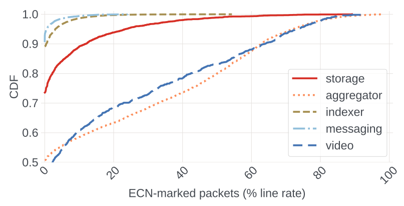
4 Understanding incast bursts

The high retransmission and ECN marking rates observed above suggest that DCTCP struggles during high-flow incast bursts. To understand these shortcomings and how to address them in future techniques, we use a simulator built atop NS3 [24] to reproduce these incasts. In Section 4.1, we examine 15 ms bursts, despite this being longer than 90% of bursts (Figure 2b), to more clearly visualize DCTCP's behavior during incast. We also present results with more common 2 ms bursts in Section 4.2. Finally, Section 4.3 examines behaviors at burst boundaries that are key to understanding the shortcomings of independent senders using window-based CCAs.

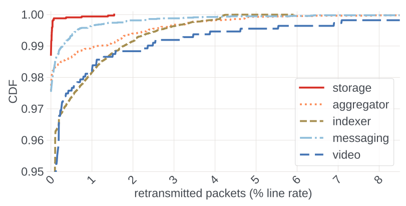
First, we briefly describe the simulation environment. N senders are connected in a dumbbell topology to a single receiver. Incast congestion occurs at the downlink on the receiver's ToR, so we primarily investigate queue length there. We configure a desired burst duration, then configure all flows with equal demand. Flows use DCTCP and their start times are jittered between 0–100 μs to model variations in processing time. Each host-ToR link is 10 Gbps and the ToR-ToR link is 100 Gbps. We examine 10 Gbps links because this creates a greater incast potential due to a 10:1 oversubscription with the inter-ToR link, whereas 400 Gbps uplinks and 100 Gbps downlinks, for example, create a smaller 4:1 oversubscription. The maximum transmission unit (MTU) is 1500 bytes. The round-trip time (RTT) is 30 μs, which is realistic for modern datacenters. The bandwidth-delay product (BDP), the amount of data required to be in flight to keep the network fully utilized, is 10 Gbps $\times$ 30 μs = 37.5 KB. We configure switch queues to have a capacity of 2 MB (1333 packets) and use an ECN marking threshold of 65 packets, as recommended by DCTCP [2]. To create the figures in this section, we simulate 11 bursts and report the average performance of the



(a) **Queuing: Incasts build long queues.**



(b) **ECN: 50% of bursts experience marks.**



(c) **Retransmits: 8% of util. can be wasted.**

Figure 4: Negative effects of incast bursts on the network. Note different y-axis ranges.

final 10 bursts. We discard the first burst because the more aggressive behavior during TCP slow start [5] results in significant packet losses and timeouts that are not representative of steady-state performance, where persistent connections are used. We disable the commonly used aggregation feature of TCP delayed ACKs [5] because it exacerbates burstiness and masks the impact of DCTCP's congestion control algorithm.

4.1 DCTCP modes of operation

We investigate how DCTCP responds to incasts of different flow counts. While these bursts are short, they do last for tens to hundreds of RTTs, and there is also high ECN signaling, so theoretically there should be sufficient time for DCTCP to reduce its congestion window. However, in practice this is only effective up to a certain number of flows (≈150 flows in this configuration). Beyond this, the queue depth is always above the ECN marking threshold because flows cannot back off any further in response to ECN marks — their CWND is at its minimum value of 1 maximum segment size (MSS). Figure 5 shows three distinct operating modes differentiated based on the queuing behavior at the receiver's ToR.

4.1.1 Mode 1: Figure 5a | 100 flows | Healthy; periodic. With 100 flows, the queue oscillates around the ECN marking threshold of 65 packets. With a BDP of 25 packets, it takes ≈90 packets in flight to trigger ECN marks. Queue depth does not approach capacity (1333 packets) and often falls below the ECN threshold, so DCTCP observes periods of no marking that trigger it to ramp up. Congestion control is functioning, so this mode is healthy. The burst completion time (BCT) is near the optimal of 15 ms. However, DCTCP still causes high queuing and loss in the production network, as shown in Section 3, even during Mode 1. First, we observe that queue overflows occur in practice because switch buffers are typically shared between ports, reducing their effective capacity, which the simulations do not model. While each port has a capacity limit, at runtime if multiple ports are sending traffic, then the limited overall buffer memory will be expended long before each queue reaches its individual limit. If the simulations modeled a shared switch buffer, the effective queue capacity would be lower and bursts would experience loss at lower flow counts. Second, we see more queue buildup at the start of the burst due to a subset of flows ramping up — we discuss this further in Section 4.3.

4.1.2 Mode 2: Figure 5b | 500 flows | Congestion control breaks down; non-periodic. With 500 flows, congestion control breaks down. Even if each flow reduces its CWND to 1 MSS (a single packet per flow), the total data in flight (500 packets) is always higher than the ECN marking threshold (≈90 packets, including the BDP). We refer to this state where all flows converge to a CWND of 1 as the *degenerate point*. Beyond this state, senders have no recourse to further control congestion — they cannot back off further — so the queue depth is simply equal to the number of flows minus the BDP. We still see a spike at the start of the burst caused by straggler flows that were able to ramp up at the end of the previous burst (Section 4.3). To generate losses in steady state, we must simply increase the flow count K until $K >$ queue length + BDP. The BCT is still near optimal, but queuing delay is now significant ($\approx 480\mu$s). We chose a deep queue (1333 packets) for illustrative purposes, but the measurements in Section 3 show that with 500 flows losses do occur in practice (e.g., due to buffer sharing).

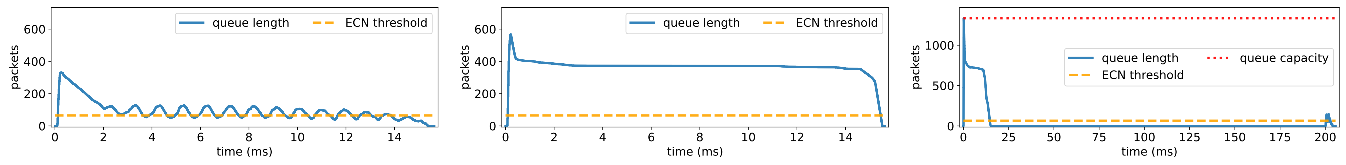
4.1.3 Mode 3: Figure 5c | 1000 flows | Timeouts. Increasing the flow count to 1000 flows results in packet losses that incur timeouts due to the small TCP windows in use. TCP's normal triple-dupACK fast retransmit [5] does not function and losses can only be detected via timeouts, which increase the BCT significantly (≈200 ms in this case). Note that the y-axis in this figure extends higher to show the overflow that causes drops. The DCTCP paper [2] notes that its algorithm breaks down during incast once there are so many flows that the first window of data is enough to cause loss, similar to the 1000-flow case described here.

4.2 Short bursts, the common case

We illustrated these modes with 15 ms bursts, but this behavior is similar during shorter bursts. Recall from Section 3.2 that 60% of bursts are 2 ms or less. Figure 6 shows the queue length over time for 2 ms bursts with various degrees of incast. Short bursts are dominated by the queuing spike caused by the flows initially transmitting an entire window at once, and there is insufficient time for the oscillatory steady state behavior shown with 15 ms bursts to occur. This higher fraction of time with deep queues combined with less reaction time before a burst completes means that this common case creates a greater challenge than long-lived bursts.

4.3 Divergence at burst boundaries

The cyclic nature of these bursts is particularly challenging for DCTCP. Persistent connections mean that congestion control state is reused from one burst to the next. Unfairness develops within the large set of flows, as follows: Some flows finish early. The stragglers then ramp up their CWND to take advantage of the available capacity at the end of the burst. At the start of the next burst, these stragglers start with a much higher CWND, dominating



(a) Mode 1: 100 flows. (b) Mode 2: 500 flows. (c) Mode 3: 1000 flows.

Figure 5: DCTCP operating modes, in terms of ToR queue length (capacity = 1333 packets). Note different axes ranges.

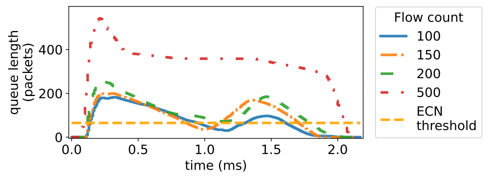


Figure 6: Queue behavior during 2 ms incast bursts.

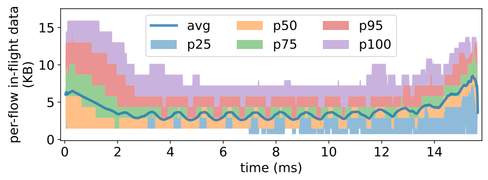


Figure 7: Per-flow in-flight data during a 100-flow incast is highly skewed. Stragglers ramp up needlessly, causing high queuing at the beginning of the next burst.

the other flows and causing a spike in queue length that can lead to loss. We see this queue spike in Figure 5, at the start of the burst. Then a new set of straggler flows develops, and the process repeats.

Figure 7 shows the distribution of per-flow in-flight data over time for the *active* flows in a 100-flow incast (Mode 1 in Section 4.1). We see a long tail (p95 and p100) of flows that transmit several times the average and median. At the end of the burst, the average per-flow in-flight data increases as the stragglers ramp up to claim newly available bandwidth. These stragglers have, in effect, *unlearned* the correct CWND for use during an incast. Even though in Mode 1 DCTCP converges to a low CWND and performs well in steady-state operation, the bursty nature of these workloads causes divergence that can build queues, leading to loss and high delay.

5 Discussion

This section reflects on key takeaways from these measurements that we believe should influence future incast solutions.

5.1 Interactions across bursts

In Mode 1 in Section 4.1, where flow count is low and each flow has many packets in flight, DCTCP performs well except for divergence between bursts that arises because stragglers *unlearn* observations made during the burst. While we focus on DCTCP, these challenges apply to independent, window-based CCAs in general. We could modify TCP behavior to explicitly "remember" such observations during incast workloads, e.g., based on the flow count distributions of past bursts. However, this may cause TCP to react less quickly to actual changes in network conditions. Another approach is to tune the CCA's parameters, such as g in DCTCP, to react more quickly to congestion, but this is brittle and does not address the root cause. Other proposals for centralized [8, 25] and receiver-based [7, 14, 16, 22] flow scheduling address incast with thousands of flows, but necessitate replacing TCP, a significant deployment hurdle. An alternative would be to provide simple guardrails that prevent TCP from ramping up excessively during incast, maintaining responsiveness but limiting TCP's ability to use available bandwidth. Such guardrails would also limit queue growth during slow start, a key source of loss even at low flow counts.

5.2 Limits to window control

Window-based TCP's rate control, loss recovery, and efficiency struggle at high flow counts that necessitate small window sizes, as shown in Modes 2 and 3 above. An alternative is Swift [19], which enables O(10k) incast by using a pacing mode that sends one packet every several RTTs. While powerful, this approach has three downsides. First, sending packets infrequently may harm receiver CPU efficiency by interfering with optimizations such as Generic Receive Offload (GRO) [10] and TCP ACK coalescing [5] that require many packets from the same flow. Second, infrequent probing robs senders of congestion feedback, so a CCA's internal model may be stale by the time it sends a packet. Consequently, pacing is useful only for long incasts: e.g., for 5000 flows, Swift presents a 20 second experiment, whereas our incast bursts complete in milliseconds. Finally, Swift is challenging to deploy because it requires software changes to both endpoints and accurate timestamps from the NIC, which behave inconsistently across NIC models. INT-based [1, 3, 20, 31] and programmable hardware [3] approaches also face deployment challenges. Instead of chasing high flow counts, an alternative approach is to divide, or *schedule*, a large incast into a series of smaller incasts where only a manageable number of flows are active at once. With fewer flows, each would operate in a healthier CWND regime, both for TCP and the receiving host. Such a scheduling technique is reminiscent of earlier work [7, 14, 16, 22], but, due to DCTCP's reasonable steady-state behavior, need only serve as an enhancement rather than a replacement to TCP.

Acknowledgments

We would like to thank Abhishek Dhamija, Rajiv Krishnamurthy, Omar Baldonado, Justine Sherry, the anonymous reviewers, and our shepherd for their support and feedback. This work was supported in part by the National Science Foundation through grant 2212390.

References

[1] Vamsi Addanki, Oliver Michel, and Stefan Schmid. 2022. PowerTCP: Pushing the Performance Limits of Datacenter Networks. In *USENIX NSDI 2022*. USENIX Association, Renton, WA, 51–70. https://www.usenix.org/conference/nsdi22/presentation/addanki

[2] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). *ACM SIGCOMM CCR* 40, 4 (Aug. 2010), 63–74. https://doi.org/10.1145/1851275.1851192

[3] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. 2023. Bolt: Sub-RTT Congestion Control for Ultra-Low Latency. In *USENIX NSDI 2023*. USENIX Association, Boston, MA, 219–236. https://www.usenix.org/conference/nsdi23/presentation/arslan

[4] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network traffic characteristics of data centers in the wild. In *ACM IMC 2010* (Melbourne, Australia). ACM, New York, NY, USA, 267–280. https://doi.org/10.1145/1879141.1879175

[5] Ethan Blanton, Vern Paxson, and Mark Allman. 2009. TCP Congestion Control. RFC 5681. https://doi.org/10.17487/RFC5681

[6] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. 2009. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking* (Barcelona, Spain) *(WREN '09)*. ACM, New York, NY, 73–82. https://doi.org/10.1145/1592681.1592693

[7] Inho Cho, Keon Jang, and Dongsu Han. 2017. Credit-Scheduled Delay-Bounded Congestion Control for Datacenters. In *ACM SIGCOMM 2017* (Los Angeles, CA). ACM, New York, NY, 239–252. https://doi.org/10.1145/3098822.3098840

[8] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. 2014. Efficient coflow scheduling with Varys. In *ACM SIGCOMM 2014* (Chicago, IL). ACM, New York, NY, 443–454. https://doi.org/10.1145/2619239.2626315

[9] Abhishek Dhamija, Balasubramanian Madhavan, Hechao Li, Jie Meng, Shrikrishna Khare, Madhavi Rao, Lawrence Brakmo, Neil Spring, Prashanth Kannan, Srikanth Sundaresan, and Soudeh Ghorbani. 2024. A large-scale deployment of DCTCP. In *USENIX NSDI 2024*. USENIX Association, Santa Clara, CA, 239–252. https://www.usenix.org/conference/nsdi24/presentation/dhamija

[10] The Linux Kernel Documentation. 2023. Segmentation Offloads. https://www.kernel.org/doc/html/latest/networking/segmentation-offloads.html.

[11] eBPF. 2022. eBPF - Introduction, Tutorials & Community Resources. https://ebpf.io.

[12] Wesley Eddy. 2022. Transmission Control Protocol (TCP). RFC 9293. https://doi.org/10.17487/RFC9293

[13] Sally Floyd, K. K. Ramakrishnan, and David L. Black. 2001. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168. https://doi.org/10.17487/RFC3168

[14] Peter X. Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2015. PHost: Distributed near-Optimal Datacenter Transport over Commodity Network Fabric. In *ACM CoNEXT 2015* (Heidelberg, Germany). ACM, New York, NY, Article 1, 12 pages. https://doi.org/10.1145/2716281.2836086

[15] Ehab Ghabashneh, Yimeng Zhao, Cristian Lumezanu, Neil Spring, Srikanth Sundaresan, and Sanjay Rao. 2022. A Microscopic View of Bursts, Buffer Contention, and Loss in Data Centers. In *ACM IMC 2022* (Nice, France). ACM, New York, NY, 567–580. https://doi.org/10.1145/3517745.3561430

[16] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-Architecting Datacenter Networks and Stacks for Low Latency and High Performance. In *ACM SIGCOMM 2017* (Los Angeles, CA). ACM, New York, NY, 29–42. https://doi.org/10.1145/3098822.3098825

[17] Raj Joshi, Ting Qu, Mun Choon Chan, Ben Leong, and Boon Thau Loo. 2018. BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks. In *ACM APSys 2018* (Jeju Island, Republic of Korea). ACM, New York, NY, USA, Article 8, 8 pages. https://doi.org/10.1145/3265723.3265731

[18] Elie Krevat, Vijay Vasudevan, Amar Phanishayee, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan. 2007. On Application-Level Approaches to Avoiding TCP Throughput Collapse in Cluster-Based Storage Systems. In *Proceedings of the 2nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07* (Reno, Nevada) *(PDSW '07)*. ACM, New York, NY, 1–4. https://doi.org/10.1145/1374596.1374598

[19] 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 *ACM SIGCOMM 2020* (Virtual Event). ACM, New York, NY, 514–528. https://doi.org/10.1145/3387514.3406591

[20] 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 *ACM SIGCOMM 2019* (Beijing, China). ACM, New York, NY, 44–58. https://doi.org/10.1145/3341302.3342085

[21] Shiyu Liu, Ahmad Ghalayini, Mohammad Alizadeh, Balaji Prabhakar, Mendel Rosenblum, and Anirudh Sivaraman. 2021. Breaking the Transience-Equilibrium Nexus: A New Approach to Datacenter Packet Transport. In *USENIX NSDI 2021*. USENIX Association, 47–63. https://www.usenix.org/conference/nsdi21/presentation/liu

[22] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities. In *ACM SIGCOMM 2018* (Budapest, Hungary). ACM, New York, NY, 221–235. https://doi.org/10.1145/3230543.3230564

[23] David Nagle, Denis Serenyi, and Abbie Matthews. 2004. The Panasas ActiveScale Storage Cluster - Delivering Scalable High Bandwidth Storage. In *SC '04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*. 53. https://doi.org/10.1109/SC.2004.57

[24] ns 3. 2020. ns-3: a discrete-event network simulator for Internet systems. https://www.nsnam.org.

[25] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2014. Fastpass: a centralized "zero-queue" datacenter network. *ACM SIGCOMM CCR* 44, 4 (Aug. 2014), 307–318. https://doi.org/10.1145/2740070.2626309

[26] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan. 2008. Measurement and Analysis of TCP Throughput Collapse in Cluster-Based Storage Systems. In *USENIX FAST 2008* (San Jose, CA). USENIX Association, USA, Article 12, 14 pages.

[27] The Linux Documentation Project. 2023. Introduction to Linux Traffic Control. https://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html.

[28] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *ACM SIGCOMM 2015* (London, United Kingdom). ACM, New York, NY, USA, 123–137. https://doi.org/10.1145/2785956.2787472

[29] Srikanth Sundaresan, Neil Spring, and Yimeng Zhao. 2023. A fine-grained network traffic analysis with Millisampler. https://engineering.fb.com/2023/04/17/networking-traffic/millisampler-network-traffic-analysis/.

[30] Balajee Vamanan, Jahangir Hasan, and T. N. Vijaykumar. 2012. Deadline-Aware Datacenter TCP (D2TCP). In *ACM SIGCOMM 2012* (Helsinki, Finland). ACM, New York, NY, 115–126. https://doi.org/10.1145/2342356.2342388

[31] Weitao Wang, Masoud Moshref, Yuliang Li, Gautam Kumar, T. S. Eugene Ng, Neal Cardwell, and Nandita Dukkipati. 2023. Poseidon: Efficient, Robust, and Practical Datacenter CC via Deployable INT. In *USENIX NSDI 2023*. USENIX Association, Boston, MA, 255–274. https://www.usenix.org/conference/nsdi23/presentation/wang-weitao

[32] Haitao Wu, Zhenqian Feng, Chuanxiong Guo, and Yongguang Zhang. 2010. ICTCP: Incast Congestion Control for TCP in Data Center Networks. In *ACM CoNEXT 2010* (Philadelphia, PA). ACM, New York, NY, Article 13, 12 pages. https://doi.org/10.1145/1921168.1921186

[33] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-resolution measurement of data center microbursts. In *ACM IMC 2017* (London, United Kingdom). ACM, New York, NY, USA, 78–85. https://doi.org/10.1145/3131365.3131375

A Artifact and data availability

The production tools and raw data in Section 3 will not be released for proprietary and privacy reasons. The simulation tools and raw data in Section 4 are available at **github.com/cmu-snap/incast-bursts**.

B Ethics

This work does not raise any ethical issues.