DIBS: Just-in-time Congestion Mitigation for Data Centers

Kyriakos Zarifis *†, Rui Miao*†, Matt Calder†, Ethan Katz-Basset†, Minlan Yu†, and Jitendra Padhye‡

†University of Southern California — ‡Microsoft Research
{kyriakos,rmiao,calderm,ethan.kb,minlanyu}@usc.edu — padhye@microsoft.com

Abstract

Data centers must support a range of workloads with differing demands. Although existing approaches handle routine traffic smoothly, intense hotspots–even if ephemeral–cause excessive packet loss and severely degrade performance. This loss occurs even though congestion is typically highly localized, with spare buffer capacity at nearby switches. In this paper, we argue that switches should share buffer capacity to effectively handle this spot congestion without the monetary hit of deploying large buffers at individual switches. Specifically, we present detour-induced buffer sharing (DIBS), a mechanism that achieves a near lossless network without requiring additional buffers at individual switches. Using DIBS, a congested switch detours packets randomly to neighboring switches to avoid dropping the packets. We implement DIBS in hardware, on software routers in a testbed, and in simulation, and we demonstrate that it reduces the 99th percentile of delay-sensitive query completion time by up to 85%, with very little impact on other traffic.

1. Introduction

Modern data center networks (DCNs) must support a range of concurrent applications with varying traffic patterns, from long-running flows that demand throughput over time to client-facing Web applications that must quickly compile results from a collection of servers. To reduce cost and queuing delay, DCN switches typically offer very shallow buffers,[1] leading to packet losses–and therefore slow transfers–under bursty traffic conditions [29, 49].

Researchers have proposed a number of solutions to tackle this problem [17, 19, 20, 24, 50–53], including DCTCP [18]. DCTCP uses ECN marking to signal congestion early to senders–before buffer overflows. This approach effectively slows long-lasting flows. However, no transport-layer congestion control scheme can reliably prevent packet loss when switch buffers are shallow and traffic bursts are severe and short-lived. One extreme example is a large number of senders each sending only a few packets in a burst to a single receiver. Since congestion control–and senders in general–cannot react in time, switches must attempt to absorb the burst. But since deep-buffered switches are expensive to build, generally switches have to drop packets–even though data center congestion is typically localized [37], meaning that the network as a whole has capacity to spare.

This extreme scenario highlights a key assumption that pervades much of the DCN work: when a switch needs to buffer a packet, it must use only its own available buffer. If its buffer is full, it must drop that packet. This assumption is so taken for granted that no one states it explicitly.

In this paper, we challenge this assumption. We advocate that, faced with extreme congestion, the switches should pool their buffers into a large virtual buffer to absorb the traffic burst, instead of dropping packets.

To share buffers among switches, we propose that a switch detour excess packets to other switches–instead of dropping them–thereby temporarily claiming space available in the other switches' buffers. We call this approach detour-induced buffer sharing (DIBS).

DIBS is easiest to explain if we assume output-buffered switches, although it can work with any switch type. When a packet arrives at a switch input port, the switch checks to see if the buffer for the destination port is full. If so, instead of dropping the packet, the switch selects one of its other ports to forward the packet on.[2] Other switches will buffer and forward the packet, and it will make its way to its destination, possibly coming back through the switch that originally detoured it.

Figure 1 shows an example. It depicts the path of a single packet (from one of our simulations) that was detoured multiple times, before reaching destination R. The weight of an arc indicates how often the packet traversed that specific arc. Dashed arcs indicate detours. While the actual order of the

* primary author

[1] Arista 7050QX-32 has just 12MB of buffer to be shared by as many as 104 ports (96x10Gbps + 8x40Gbps). This is because the high-speed memory that forms the buffer must support NxC read-write bandwidth, where N is the number of ports and C is the nominal link speed. The cost of memory increases as N and C increase.

[2] We will describe later how we select the port. Specifically, we avoid ports whose buffers are full, and also those that are connected to end hosts.

EuroSys '14, April 13–16, 2014, Amsterdam, Netherlands.
Copyright © 2014 ACM 978-1-4503-2704-6/14/04…$15.00.
http://dx.doi.org/10.1145/2592798.2592806

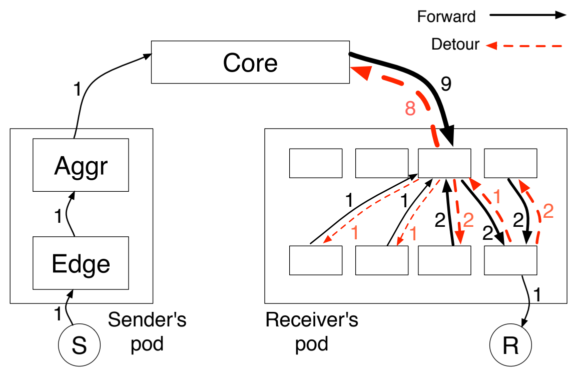


Figure 1: Example path of a packet detoured 15 times in a K=8 fat-tree topology. For simplicity, we only show 1 edge switch and 1 aggregation switch in the sender's pod, and abstract the 16 core switches into a single node. The numbers the arc thicknesses indicate how often the packet traversed that arc.

hops cannot be inferred, we can see that the packet bounced 8 times back to a core switch because the aggregation switch was congested. The packet bounced several times within the receiver pod prior to reaching the receiver.

The idea of detouring–and not dropping–excess traffic seems like an invitation to congestion collapse [29]. Our key insight is to separate the slowing of senders to avoid congestion collapse from the handling of excess traffic necessary before congestion control kicks in. Our results show that DIBS works well as long as a higher-layer congestion control scheme such as DCTCP suppresses persistent congestion and the network has some spare capacity for DIBS to absorb transient congestion. We will formalize these requirements later in the paper and show that they are easily satisfied in a modern DCN.

In fact, DIBS is particularly suited for deployment in DCNs. Many popular DCN topologies offer multiple paths [16], which detouring can effectively leverage. The link bandwidths in DCNs are high and the link delays are small. Thus, the additional delay of a detour is low. DCNs are under a single administrative control, so we do not need to provide incentives to other switches to share their buffer.

DIBS has two other key benefits. First, DIBS does not come into play until there is extreme congestion–it has no impact whatsoever when things are "normal". Second, the *random detouring* strategy we propose in this paper has no parameters to tune, which makes implementation very easy.

In the rest of the paper, we will describe the DIBS idea in detail, and evaluate DIBS extensively using a NetFPGA [8] implemenation, a Click [38] implementation and simulations using NS-3 [9]. Our results show that *DIBS significantly improves query completion time when faced with heavy congestion*. In cases of heavy congestion, DIBS can reduce 99th percentile of query completion times by as much as 85%. Furthermore, this improvement in performance of query traffic is achieved with *little or no collateral damage to background traffic.* for typical data center traffic patterns. We also investigate how extreme the traffic must be before DIBS "breaks", and find that DIBS handle traffic load of up to 10000 queries per second in our setting. Finally, we compare the performance of DIBS to pFabric [20] (the state-of-the-art datacenter transport designs intended to achieve near-optimal flow completion times). We find that during heavy congestion, DIBS performs as well (if not slightly better) in query completion time while having less impact on the background traffic.

2. DIBS overview

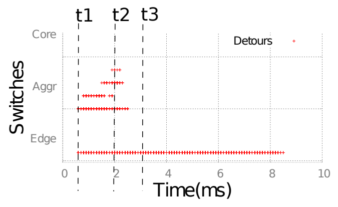
DIBS is not a congestion control scheme; indeed, it must be paired with congestion control (§3). Nor is DIBS a completely new routing scheme, relying on the underlying Ethernet routing (§3). Instead, DIBS is a small change to that normal Ethernet (L2) routing. In today's data centers, when a switch receives more traffic than it can forward on a port, it queues packets at the buffer for that port.[3] If the buffer is full, the switch drops packets. With DIBS, the switch instead forwards excess traffic via other ports.

When a detoured packet reaches the neighboring switch, it may forward the packet towards its destination normally, or, if it too is congested, it may detour the packet again to one of its own neighbors. The detoured packets could return to the original switch before being forwarded to its destination, or it could reach the destination using a different path.

Single packet example. Figure 1 depicts the observed path of a single packet that switches detoured 15 times on its way to its destination *R*. This example came from an trace in our simulation of a K=8 fat-tree topology with a mixture of long flows and short bursts, modeled on actual production data center workloads [18]. We discuss our simulations more in §5.3. Here, we just illustrate how DIBS moves excess traffic through the network.

When the packet first reached an aggregation switch in *R*'s pod, the switch's buffer on its port toward *R* was congested, so the aggregation switch detoured the packet. In fact, most of the detouring occurred at this switch. To avoid dropping the packet, the switch detoured the packet to other edge switches four times and back to core switches eight times, each time receiving it back. After receiving the packet back the twelfth time, the switch had buffer capacity to enqueue the packet for delivery to *R's* edge switch. However, the link from that switch to *R* was congested, and so the edge switch detoured the packet back to the aggregation layer three times. After the packet returned from the final detour, the edge switch finally had the buffer capacity to deliver the packet to *R*. The path of this packet illustrates how DIBS effectively leverages neighboring switches to buffer the packet, keeping it close unless congestion subsides, rather than dropping it.

[3] We assume an output queued architecture for the ease of description. DIBS can be implemented in a variety of switch architectures (§4).



(a) Detours per switch over time. (Each dot denotes the decision of a switch to detour a packet.)

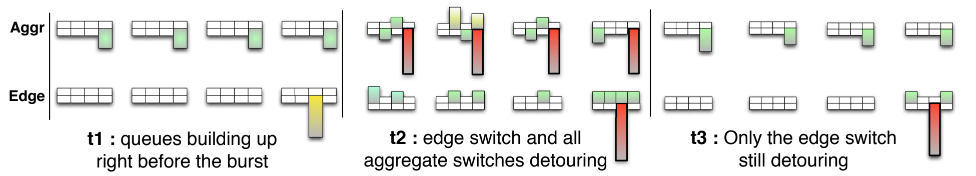

Aggr
Edge
t1 : queues building up right before the burst
t2 : edge switch and all aggregate switches detouring
t3 : Only the edge switch still detouring
(b) Buffer occupancy at times t1, t2, t3 in a pod. Each switch is represented by 8 bars. Each bar is an outgoing port connecting to a node in the layer below or above. The size of the bar represents the port's output queue length: (green: packets in buffer; yellow: buffer buildup; red: buffer overflow).

Figure 2: Detours and buffer occupancy of switches in a congested pod. During heavy congestion, multiple switches are detouring.

Network-wide example. In the next example, a large number of senders send to a single destination, causing congestion at all the aggregate switches in a pod, as well as at the destination's edge switch. Figure 2(a) illustrates how the switches respond to the congestion over time, with each horizontal slice representing a single switch and each vertical slide representing a moment in time. Each marker on the graph represents a switch detouring a single packet. From time $t1$ until just after $t2$, four aggregation switches have to detour a number of packets, and the edge switch leading to the destination has to detour over a longer time period. Even with this period of congestion, DIBS absorbs and delivers the bursts within 10ms, without packet losses or timeouts.

Figure 2(b) shows the buffer occupancy of the eight switches in the destination's pod over the first few milliseconds of the bursts. The bursts begin at time $t1$, with the aggregation switches buffering some packets to deliver to the destination's edge switch and the edge switch buffering a larger number of packets to deliver to the destination. Soon after, at time $t2$, all five of those buffers are full, and the five switches have to detour packets. The figure depicts the destination's edge switch buffering packets randomly to detour back to each of the aggregation switches, and the aggregation switches scheduling packets to detour to the other edge switches and back to the core. By time $t3$, most of the congestion has abated, with only only the edge switch of the receiver needing to continue to detour packets.

These examples illustrate how DIBS shares buffers among switches to absorb large traffic bursts. They also highlight the four key decisions that DIBS needs to make: (i) when to start detouring; (ii) which packets to detour; (iii) where to detour them to; and (iv) when to stop detouring. By answering these questions in different ways, we can come up with a variety of detouring policies.

In this paper, we focus on the simplest policy. When a switch receives a packet, if the buffer towards that packet's destination is full, the switch detours the packet via a random port that is connected to another switch,[4] and whose buffer is not full. In §8, we briefly consider other detouring policies.

We conclude by noting two salient features of DIBS.

[4] We do not detour packets to end hosts, because end hosts do not forward packets not meant for them.

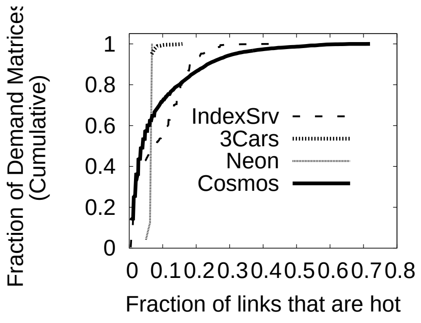


Figure 3: Sparsity of hotspots in four workloads

DIBS has no impact on normal operations. It comes into play only in case of severe congestion. Otherwise, it has no impact on normal switch functioning.

DIBS has no tunable parameters. The random detouring strategy does not have any parameters that need tuning, which makes it easy to implement and deploy. It also does have not any coordination between switches, and thus detouring decisions can be made at line rate.

3. Requirements

DIBS can improve performance only if certain conditions are met. We now describe these requirements, and show that they are easily satisfied in modern DCNs.

No persistent congestion: DIBS is based on the premise that even though a single switch may be experiencing congestion and running out of buffer space, other switches in the network have spare buffer capacity. We also assume that the network has sufficient bandwidth to carry detoured packets to the switches with buffer capacity. In other words, DIBS requires that congestion in a data center environment is rare, localized, and transient. Several studies of traffic patterns in data centers satisfy this requirement [36, 37]. In fact, the Flyways system also relies on this observation [37].

Figure 3 reproduces a graph from the Flyways paper [37]. It shows measurements from four real-world data center workloads. The data sets represent different traffic patterns, including map-reduce, partition-aggregate, and high performance computing. The graph shows the distribution (over time) of the fraction of links that are running "hot," with utilization at least half that of the most loaded link. At least 60% of the time in every dataset, fewer than 10% of links are running hot.

We see similar results when we use scaled versions of workload from [18] in our simulations (§5). Figure 4 shows the fraction of links with utilization of 90% or more for three different levels of traffic (using default settings in Table 2 except qps). We use 90% as a threshold, since it is more representative of the extreme congestion that DIBs is designed to tackle[5]. The takeaway remains the same – at any time, only handful of the links in the network are "hot". Figure 5 shows that for both baseline and heavy cases, there is plenty of spare buffer in the neighborhood of a hot link. Although the heavy workload induces 3X more hot links than the baseline does, the fraction of available buffers in nearby switches are just slightly reduced. In fact, we see that nearly 80% of the buffers on switches near a congested switch are empty in all cases except the extreme scenario where dibs fails (§5).

Congestion control: DIBS is meant for relieving short-term congestion by sharing buffers between switches. It is not a replacement for congestion control. To work effectively, DIBS must be paired with a congestion control scheme.

The need for a separate congestion control scheme stems from the fact that DIBS does nothing to slow down senders. Unless the senders slow down, detoured packets will eventually build large queues everywhere in the network, leading to congestion collapse. To prevent this, some other mechanism must signal the onset of congestion to the senders.

Because DIBS is trying to avoid packet losses, the congestion control mechanism used with it cannot rely on packet losses to infer congestion. For example, we cannot use TCP NewReno [14] as a congestion control mechanism along with DIBS. Since NewReno slows down only when faced with packet losses, the senders may not slow down until all buffers in the network are filled, and DIBS is forced to drop a packet. This not only defeats the original purpose of DIBS, but also results in unnecessary and excessive queue buildup.

In this paper, we couple DIBS with DCTCP [18], which uses ECN marking [5] instead of packet losses to signal congestion.

No spanning-tree routing: Enterprise Ethernet networks use spanning tree routing. Detouring packets on such a network would interfere with the routing protocol itself. On the other hand, data center networks typically offer multiple paths and hence do not rely on spanning trees. Following recent data center architecture designs [16, 32, 44], we assume that switches forward packets based on forwarding tables (also known as FIBs). A centralized controller may compute the FIBs, or individual switches may compute them using distributed routing protocols like OSPF or ISIS. When there are multiple shortest paths available, a switch uses flow-level equal-cost multi-path (ECMP) routing to pick the outgoing

[5] With threshold set to 50%, the graph looks similar to Figure 3.

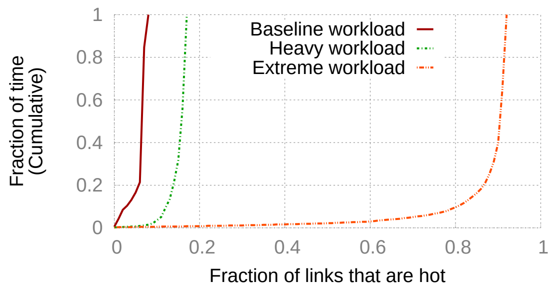


Figure 4: Hotlinks. Baseline workload is 300 qps, high is 2000 qps and extreme is 10,000 qps. See Table 2.

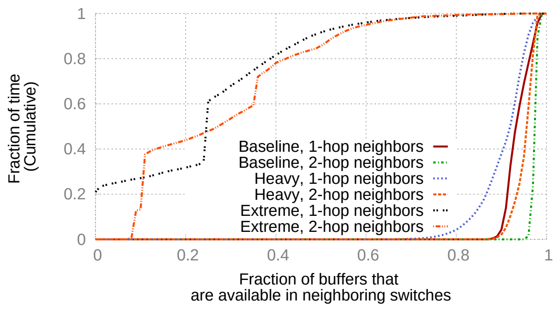


Figure 5: Neighboring buffer size

port for each flow [35]. We assume that switches do not invalidate entries in their FIB if they happen to detect loops.

4. Design Considerations and Implications

We now discuss several issues that stem from the DIBS design and requirements.

Interactions with TCP: By detouring some packets along longer paths, DIBS can affect TCP in three ways. First, DIBS can reorder packets within a flow. Such reordering is rare, since DIBS detours packets only in cases of extreme congestion. Reordering can be dealt with either by disabling fast retransmissions entirely [20] or by increasing the number of duplicate acknowledgments required to trigger them [54]. For all the experiments presented in the paper that used DIBS, we disabled fast retransmissions. We have also experimented with simply increasing the dupack threshold, and found that a dup-ack threshold of larger than 10 packets is usually sufficient to deal with reordering caused by DIBS. Other than this, no changes are required on the hosts.

Second, packets traveling along longer paths can inflate TCP's estimate of round-trip time (RTT) and RTT variance, which TCP uses to calculate a retransmission timeout (RTO) value to detect loss. However, packet losses are exceedingly rare with DIBS, so the value of the timeout is not important. Indeed, we do not even need to set MinRTO to a very low value, as advocated by some prior proposals [29]. Instead, we use a default MinRTO value of 1ms, which is commonly used in data center variants of TCP [18].

Third, excessive detouring delays may trigger a spurious retransmission by the sender. Since we do not need to set an aggressive MinRTO, spurious retransmissions are rare.

Loops and multiple detours: Detour paths may take packets along loops on their way to the destination. For example, in Figure 1, the packet loops between the receiver's edge switch and an aggregation switch twice, among other loops. Loops can complicate network management and troubleshooting, and so operators and routing protocols generally try to avoid them. We believe that the loops induced by DIBS represent a net win, because DIBS reduces loss and flow completion times, and the loops are transient and only occur during periods of extreme congestion. In §5, we show that DIBS only requires a limited number of detours and only causes loops in a transient time period.

Collateral damage: DIBS may cause "collateral damage." For instance, if the example in Figure 2 had additional flows, the detoured packets may have gotten in their way, increasing their queuing delay and, possibly, causing switches to detour packets from the other flows. However, we will show in §5 that such collateral damage is uncommon and limited for today's data center traffic. This is because DIBS detours packets only in rare cases of extreme congestion. Also, in the absence of DIBS, some of these flows may have performed *worse*, due to packet drops caused by buffer overflows at the site of congestion. We will see examples of such traffic patterns in §5.

Switch buffer management: So far, our description of DIBS has assumed that the switch has dedicated buffers per output port. Many switches in data centers have a single, shallow packet buffer, shared by all ports. The operator can either statically partition the buffer among output ports, or configure the switch to dynamically partition it (with a minimum buffer on each port to avoid deadlocks). DIBS can be easily implemented on both types of switches. With DIBS, if the switch reaches the queuing threshold for one of the output ports, it will detour the packets to a randomly selected port. §5.5.2 evaluates DIBS with dynamic buffer allocations.

Switches can also have a combined Input/Output queue (CIOQ) [53]. Since these switches have dedicated egress queues, we can easily implement DIBS in this architecture. When a packet arrives at an input port, the forwarding engine determines its output port. If the desired output queue is full, the forwarding engine can detour the packet to another output port.

Comparison to ECMP and Ethernet flow control: Detouring is closely related to Ethernet flow control [31] and ECMP [35]. We provide a more detailed comparison in §6.

5. Evaluation

We have implemented DIBS in a NetFPGA switch, in a Click modular router [38] and in the NS3 [10] simulator. We use these implementations to evaluate DIBS in increasingly sophisticated ways. Our NetFPGA implementation validates that DIBS can be implemented at line rate in today's switches (§5.1). We use our Click implementation for a small-scaled testbed evaluation (§5.2).

We conduct the bulk of our evaluation using NS-3 simulations, driven by production data center traffic traces reported in [18]. Unless otherwise mentioned, we couple DIBS with DCTCP and use the random detouring strategy (§2). The simulations demonstrate that, across a wide range of traffic (§5.4) and network/switch configurations (§5.5), DIBS speeds the completion of latency-sensitive traffic without unduly impacting background flows, while fairly sharing bandwidth between flows (§5.6). Of course, DIBS only works if the congestion control scheme it is coupled with is able to maintain buffer capacity in the network; we demonstrate that extremely high rates of small flows can overwhelm the combination of DCTCP and DIBS (§5.7). Finally, we show that DIBS can even outperform state-of-the-art datacenter transport designs intended to achieve near-optimal flow completion times [20] (§5.8).

5.1 NetFPGA implementation and evaluation:

To demonstrate that we can build a DIBS switch in hardware, we implemented one on a 1G NetFPGA [8] platform. We found that it adds negligible processing overhead.

We followed the reference Ethernet switch design in NetFPGA, which implements the main stages of packet processing as pipelined modules. This design allows new features to be added with relatively small effort. We implemented DIBS in the Output Port Lookup module, where the switch decides which output queue to forward a packet to. We provide the destination-based Lookup module with a bitmap of available output ports whose queues are not full. It performs a bitwise *AND* of this bitmap and the bitmap of the desired output ports in a forwarding entry. If the queue for the desired output port is not full, it stores the packet in that queue. Otherwise, it detours the packet to an available port using the bitmap.

Our DIBS implementation requires about 50 lines of code and adds little additional logic (2 Slices, 10 Flip-Flops and 3 input Look-Up Tables). Given the response from the lookup table, DIBS decides to forward or detour within the same system clock cycle. That means DIBS does not add processing delay. Our followup tests verified that our DIBS switch can forward and detour a stream of back-to-back 64-byte packets at line-rate. Implementing DIBS requires very little change in existing hardware.

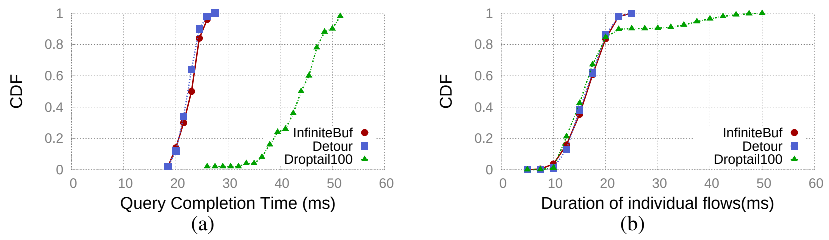


Figure 6: Click implementation of DIBS achieves near optimal Query Completion Times because no flow experiences timeouts

DC settings	Value	TCP settings	Value
Link rate	1 Gbps	minRTO	10 ms
Switch buffer	100 pkt per port	Init. cong. win.	10
MTU	1500 Bytes	Fast retransmit	disabled

Table 1: Default DIBS settings following common practice in data centers (e.g., [18]) unless otherwise specified. In Table 2, we indicate how we explore varying buffer sizes and other traffic and network parameters.

5.2 Click implementation and evaluation

We implemented the *detour* element in the Click modular router with just 50 additional lines of code. When a packet arrives at a switch, Click first performs a forwarding table lookup. Before enqueuing the packet to the appropriate output buffer, our *detour* element checks if the queue is full. If it is, the *detour* element picks a different output queue at random.

We evaluated our Click-based prototype on Emulab [4]. Our testbed was a small FatTree topology with two aggregator switches, three edge switches, and two servers per rack. We set parameters based on common data center environments as shown in Table 1. To avoid having out-of-order detoured packets cause unnecessary retransmissions, when using DIBS, we disabled fast retransmission.

We evaluated DIBS with the partition-aggregate traffic (incast) pattern known to cause incast [29]. This traffic pattern arises when a host queries multiple hosts in the network in parallel, and they all respond at the same time.

Detouring is a natural solution for incast, because it automatically and immediately increases the usable buffer capacity when the incast traffic overwhelms a switch's own buffer. We will show that detouring can achieve near optimal performance for incast traffic patterns.

In our test, the first five servers each sent ten simultaneous flows of 32KB each to the last server.[6] The primary metric of interest is query completion time (QCT), which is the time needed for the receiver to successfully receive all flows. We ran the experiment 50 times with these settings. We also tried several different settings for flow sizes, numbers of flows, and queue sizes. All incast scenarios caused qualitatively similar results.

[6] To ensure that the servers generated flows simultaneously, we modified iperf [7] to pre-establish TCP connections.

We consider three settings. In the first setting, the queue on each switch port is allowed to grow infinitely. This setting allows the switches to absorb incast burst of any size, without packet loss so it represents a useful baseline. In the second setting, we limit the queue size to 100 packets, and enforce droptail queuing discipline. The third setting also uses 100 packets buffers, but enables DIBS.

The performance of the three settings is shown in Figure 6(a). With infinite buffer, all queries complete in 25ms. DIBS also provides very similar results, with all queries completing in 27ms. QCT is much higher with the smaller buffer with droptail queuing, ranging from 26ms to 51ms.

The reason for this high QCT with droptail queuing and why DIBS provides such an improvement to it becomes apparent by observing the completion times of all the individual query flows. In Figure 6(b), in the droptail setting we notice that a small number of responses (about 9%) have durations between 25-50ms. The reason is that due to the bursty nature of the traffic, those responses suffer from packet loss, which forces them to take a timeout. In our experiment, all queries had at least one such delayed response. Since a query is not completed until all the responses have arrived, those delayed responses determine the QCT. DIBS eliminates packet drops and consequent retransmission timeouts, guaranteeing that all responses have arrived within 25ms, which is close to the optimal case. This explains the significant improvement in QCT.

5.3 Simulations setup

Now we use NS3 simulations to study the performance of DIBS on larger networks with realistic workloads. We will primarily focus on comparing the performance of DCTCP with and without DIBS, although we will also consider pFabric [20] in §5.8.

DCTCP: DCTCP is the state of the art scheme for congestion control in data center environment [18], and is currently being deployed in several large data centers supporting a large search engine. DCTCP is a combination of a RED-like [12] AQM scheme with some changes to end-host TCP stacks. The switch marks packets by setting ECN codepoints [5], once the queue exceeds a certain threshold. If the queue overflows, packets are dropped. The receiver returns

Setting	Min	Max	Section
BG inter-arrival (ms)	10	**120**	5.4.1
QPS	**300**	15000	5.4.2, 5.7
Response size (KB)	**20**	160	5.4.3, 5.7
Incast degree	**40**	100	5.4.4, 5.5.2
Buffer (packets)	20	**100**	5.5.1, 5.5.2
TTL	12	**255**	5.5.3
Oversubscription	**1**	16	5.5.4

Table 2: Simulation parameter ranges we explore. **Bold values** indicate default settings. The top half of the table captures traffic intensity, including background traffic (top row, with lower inter-arrival time indicating more traffic) and query intensity (next three rows). The bottom half of the table captures network and switch configuration, including buffer sizes, initial packet TTLs, and link oversubscription. We revisit some aspects in additional sections on shared switch buffers (§5.5.2) and extreme congestion (§5.7).

these marks to the sender. The TCP sender reduces its rate in proportion to the received marks.

DCTCP augmented with DIBS: Instead of dropping packets when the queue overflows, we detour them to nearby switches as described in §2. The detoured packets are also marked. In addition, we also disable fast retransmit on TCP senders. For brevity, we refer to this scheme simply as DIBS in this section.

To ensure fair comparison, we use settings similar to those used to evaluate DCTCP [18].

Network and traffic settings: Table 1 shows network settings. The switches use Equal-Cost Multi-Path (ECMP) routing to split flows among shortest paths. In our default configuration, we use a fixed 100 packet FIFO buffer per output port, and set the marking threshold to 20 packets. We simulate a network consisting of 128 servers, connected using a fat-tree topology ($K = 8$) and 1Gbps links.

We use the traffic distribution data from a production data center [18] to drive the simulations. We vary various parameters of these distributions, as well as switch and network configurations, to evaluate a wide range of scenarios (i.e. a "parameter sweep"). Table 2 depicts the default value of each parameter in bold (corresponding to the normal case in an earlier study [18]), the range we vary each parameter over, and the sections in which we vary each parameter.

The traffic includes both query traffic (a.k.a. partition / aggregate, a.k.a. incast) [18], and background traffic. The background traffic has 80% of flows smaller than 100KB. We vary the intensity of background traffic by varying the interarrival time. In the query traffic, each query consists of a single incast *target* that receives flows from a set of *responding nodes*, all selected at random. We vary three parameters of the query traffic to change its intensity: the degree of incast (i.e., the number of responding nodes), the size of each response, and the arrival rate of the queries. The degree of incast and the response size determine how intense a given burst will be, while the arrival rate determines how many nodes will be the target of incast in a given time period. We also vary the size of each output buffer at the switches.

In general, our intention is to start from the traffic patterns explored in [18], and consider more intense traffic by varying interarrival time of background traffic and intensity of query traffic.

Metric: The metric of interest for query traffic is 99th percentile of query completion time (QCT) [18]. This is the standard metric for measuring performance of incast-like traffic. For background traffic, the metric is flow completion time (FCT) of short flows (1-10KB). Recall that the initial window size for TCP connections in a data center setting is 10 packets (Table 1). Since these flows can complete in one window, the FCT of these flows stems almost entirely from the queueing delay experienced by these packets, as well as the delay of any loss recovery induced by congestion. We focus on the 99th percentile of FCT for these flows, to highlight any collateral damage. In contrast, DIBS has little impact on the performamce of long flows, since their throughput is primarily determined by network bandwidth.

5.4 Performance under different traffic conditions

5.4.1 Impact of background traffic

In our first experiment, we hold the query traffic constant at the default values from Table 2 and vary the inter-arrival time of the background traffic from 10ms to 120ms. The amount of background traffic *decreases* as inter-arrival time *increases*. Figure 7 shows the result. Although depicted on one graph, QCT and background FCT are separate metrics on different traffic and cannot be compared to each other.

DIBS significantly reduces the response time of query traffic. The 99th percentile of QCT is reduced by as much as 20ms. Furthermore, the improvement in QCT comes at very little cost to background traffic. The 99th percentile of background FCT increases by less than 2ms, while most of the background flows are barely affected. In other words, there is very little collateral damage in this scenario. Furthermore, we see that the collateral damage does not depend on the intensity of the background traffic.

There are two reasons for this lack of collateral damage. First, in all the experiments, on average, DIBS detours less than 20% of the packets. Over 90% of these detoured packets belong to query traffic, since DIBS detours only in cases of extreme congestion, and only query traffic causes such congestion. DIBS detours only 1% of the packets from background flows. These background packets come from flows that happen to traverse a switch where query traffic is causing a temporary buffer overflow. Without DIBS, the switch would have dropped these packets. DIBS does not suffer any packet drops in this scenario, whereas standard DCTCP does. Second, most background flows are short, and the congestion caused by an incast burst is ephemeral. Thus, most background flows never encounter an incast burst, and so it

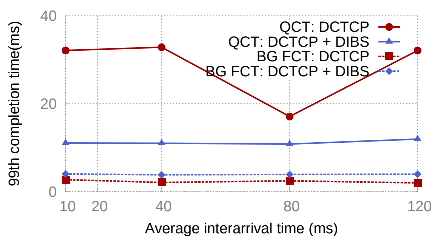


Figure 7: Variable background traffic. Collateral damage is consistently low. (incast degree: 40; response size: 20KB; query arrival rate: 300 qps). Although depicted on one graph, QCT and background FCT are separate metrics on different traffic and cannot be compared to each other.

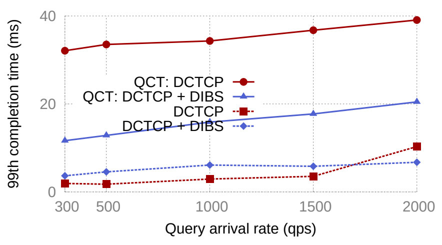


Figure 8: Variable query arrival rate. Collateral damage is low. Query traffic rate has little impact on collateral damage, and, at high query rate, DIBS improves performance of background traffic. (Background inter-arrival time: 120ms; incast degree: 40; response size: 20KB)

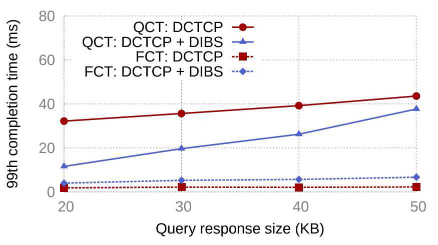


Figure 9: Variable response size. Collateral damage is low, but depends on response size. Improvement in QCT decreases with increasing response size. (Background inter-arrival time: 120ms; incast degree: 40; query arrival rate: 300 qps)

is rare for a background packet to end up queued behind a detoured packet.

5.4.2 Impact of query arrival rate

We now assess how DIBS affects background traffic as we vary the rate of queries (incasts). Keeping all other parameters at their default values (Table 2), we vary the query arrival rate from 300 per second to 2000 per second. This scenario corresponds to a search engine receiving queries at a high rate. The results are shown in Figure 8.

We see that DIBS consistently improves performance of query traffic, without significantly hurting the performance of background traffic. The 99th percentile of QCT improves by 20ms. The 99th percentile FCT shows a small increase (1-2ms). However, at the highest query arrival rate (2000 qps), DIBS improves the 99th percentile FCT for short background flows. At such high query volume, DCTCP fails to limit queue buildup. As a result, without DIBS, some background flows suffer packet loses, slowing their completions. With DIBS, background traffic does not suffer packet losses, as the packets will instead detour to other switches.

Collateral damage remains low because, even with 2000 qps, over 80% of packets are not detoured. More than 99% of the detoured packets belong to query traffic in all cases. DIBS does not suffer any packet drops in this scenario.

5.4.3 Impact of query response size

We now vary the intensity of query traffic by varying the response size from 20KB to 50KB, while keeping all other parameters at their default values (Table 2). Larger responses correspond to collecting richer results to send to a client. The results are shown in Figure 9.

We see that DIBS improves 99th percentile of QCT, but, as the query size grows, DIBS is less effective. At a response size of 20KB, the difference in 99th percentile QCT is 21ms, but, at a response size of 50KB, it is only 6ms. This behavior is expected. As the response size grows, the size of the burst becomes larger. While there are no packet drops, as more packets suffer detours, the possibility that a flow times out before the detoured packets are delivered goes up. In other words, with DIBS, TCP flows involved in query traffic suffer from spurious timeouts.

Needless to say, with DCTCP alone, there are plenty of packet drops, and both background and query traffic TCP flows routinely suffer from (not spurious) timeouts. In contrast, DIBS has no packet drops.

As before, the impact on background traffic is small, but it increases slightly with the increase in response size. The difference in the 99th percentile of FCT of short flows is 1.2ms at 20KB, while it increases to 4.4ms at 50KB. This is because more packets are detoured for a given incast burst.

5.4.4 Impact of incast degree

We now vary the intensity of incast by varying the incast degree; i.e., the number of responders involved, from 40 to 100, while keeping all other parameters at their default values (Table 2). More responders correspond to collecting more results to send to a client. Figure 10 presents the results.

We see that DIBS improves 99th percentile of QCT, and the improvement grows with increasing incast degree. For example, the difference between the two schemes is 22ms when the degree of incast is 40, but grows to 33ms when the incast degree is 100. The impact on background traffic is small, but it increases with increasing incast degree.

It is interesting to compare Figures 9 and 10, especially at the extreme end. For the extreme points in the two figures, the total amount of data transmitted in response to a query is the same (2MB). Yet, both DCTCP and DIBS perform worse when the large responses are due to a high incast degree (many senders) than when they are due to large response sizes. The drop in DCTCP's performance is far worse. With large responses, the 99th percentile QCT of DCTCP was 44ms. With many senders, it is 79ms. For DIBS, the corresponding figures are 37ms and 46ms, respectively.

The reason is that the traffic with high incast degree is far more bursty. When we explored large responses, 40 flows

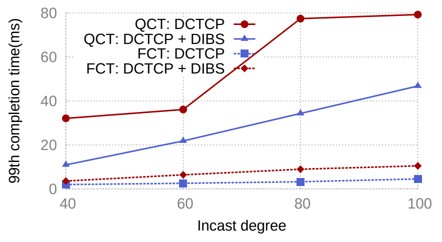
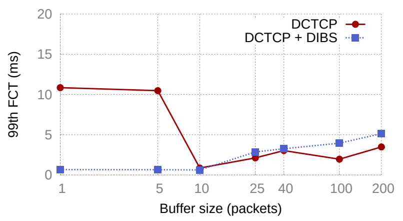
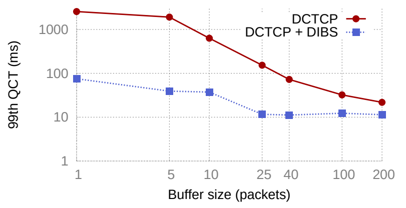


Figure 10: Variable incast degree. Collateral damage is low, but depends on incast degree. (Background inter-arrival time: 120ms; query arrival rate: 300 qps; response size: 20KB)



(a) Background traffic



(b) Query traffic

Figure 11: Variable buffer size. There is no collateral damage and DIBS performs best with medium buffer size. (Background inter-arrival rate: 10ms; incast degree: 40; response size: 20KB; query arrival rate: 300 qps)

transferred 50KB each. In the first RTT, each flow sends 10 packets (initial congestion window size). Thus, the size of the initial burst is 400KB. In contrast, with extremely high incast degree, 100 flows transfer 20KB each. Thus, the size of the first burst is 1MB.

The bursty traffic affects DCTCP far more than it impacts DIBS. With DCTCP alone, as the degree of incast grows, more and more flows belonging to the incast suffer from packet drops and timeouts.

DIBS is able to spread the traffic burst to nearby switches and ensure no packet loss. However, DIBS is not completely immune to burstiness. Packets suffer many more detours in this setting than they do in the previous setting, for a comparable total response size. For example, when the incast degree is 100, 1% of the packets are detoured 40 times or more. In contrast, in the previous setting, when the burst size was 50KB, the worst 1% of the packets suffered only about 10 detours.

5.5 Performance of different network configurations

5.5.1 Impact of buffer size

All previous experiments were carried out with buffer size of 100 packets per port. We now consider the impact of switch buffer size on DIBS performance. We vary the buffer size at the switch from 1 packet to 200 packets, while keeping all other parameters at their default values (Table 2). The results are shown in Figure 11. We show the query and background traffic results separately for better visualization.

We see that DIBS improves 99th percentile of QCT significantly at lower buffer sizes. The performance boost is more obvious at lower buffer sizes, where DCTCP suffers from more packet drops while DIBS is able to absorb the burst by spreading it between switches. This result also shows that our choice of buffer size of 100 for all past experiments is a conservative one in order to compare with the default DCTCP setting [18]. For smaller buffer sizes the performance boost of DIBS becomes more obvious.[7]

[7] The fluctuation in the figure is caused by the queries in the 99th having timeouts.

5.5.2 Impact of shared buffers

The switches deployed in production data centers usually use Dynamic Buffer Allocation (DBA) to buffer the extra packets in shared packet memory upon congestion. In our simulation, we model a DBA-capable switch with 8X1GbE ports and 1.7MB of shared memory based on the Arista 7050QX-32 switch [3] .

We use the default settings from Table 2 but vary the incast degree and compare the result with Figure 10. By enabling DBA, DCTCP has zero packet loss and DIBS is not triggered. However, when we further increase the incast degree beyond 150 by using multiple connections on single server, we find that DCTCP with DBA experiences packet loss and an increased QCT in 99th percentile. However, when DIBS is enabled, we observe no packet loss even upon extreme congestion which overflows the whole shared buffer, which leads to a decrease of 75.4% for the 99th percentile QCT.

5.5.3 Impact of TTL

So far, we have seen that the impact of DIBS on background flows was small in all but the most extreme cases. Still, it is worth considering whether we could reduce the impact further. In some cases, packets are detoured as many as 20 times. One possible way to limit the impact of detoured packets on background flows is to limit the number of detours a packet can take. This limits the amount of "interference" it can cause. The simplest way to limit detours is to restrict the TTL on the packet.

We now carry out an experiment where we vary the max TTL limit from 12 to 48, while keeping all other parameters at their default values (Table 2). Recall that the diameter of our test network is 6 hops. Each backwards detour reduces the TTL by two before the packet is delivered to the destination (e.g. one when packet is detoured, and one when it re-traces that hop). Thus when the max TTL is 12, the packet can be detoured backwards 3 times. When the max TTL is 48, the packet can be detoured backwards 9 times. "Forward" detours change the accounting somewhat.

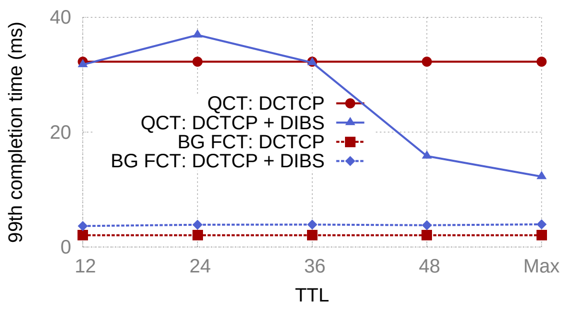


Figure 12: Variable max TTL. Limiting TTL does not have significant impact on background traffic. (Background inter-arrival rate: 10ms; incast degree: 40; response size: 20KB; query arrival rate: 300 qps)

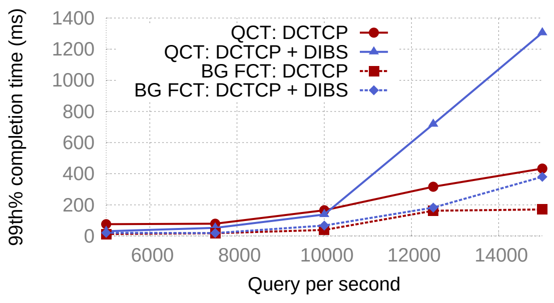


Figure 13: Extreme query intensity (Background inter-arrival time: 120ms; incast degree: 40)

Figure 12 shows the results. Note that the last point on the X axis represents maximum TTL, which is 255. We have also shown performance of DCTCP for reference. TTL values have no impact on DCTCP's performance. For DIBS, the 99th percentile QCT improves with increasing TTL. The reason is that at lower TTL, DIBS is forced to drop packets due to TTL expiration. We also see that TTL has little impact on 99th percentile FCT of background flows.

We note one other interesting observation. Note that DIBS performs slightly better with TTL of 12, instead of TTL of 24. We believe that with TTL of 24, packets stay in the network for far too long, only to get dropped. In other words, we waste resources. Thus, it may be best not to drop a packet that's been detoured many times! We are investigating this anomaly in more detail.

We have carried out this experiments in other settings (e.g. higher incast degree etc.) and have found the results to be qualitatively similar.

5.5.4 Impact of oversubscription

In order to study the effect of oversubscribed links on the performance of DIBS, we repeated the previous experiments with different oversubscription parameters for the fat-tree topology. This was done by lowering the capacity of the links between switches by a factor of 2, 3 and 4 (providing oversubscription of 1:4, 1:9 and 1:16 respectively). Our experiments showed that DIBS consistently lowers the 99th percentile QCTs by 20ms, for every oversubscription setting, without affecting the background FCTs. This is because an increasing number of packets are buffered in the upstream path when oversubscription factor increases, but the last hop of the downstream path is still the bottleneck for query traffic. So at the last hop, DIBS can avoid packet loss and fully utilize bottleneck bandwidth.

5.6 Fairness

We now show that DIBS ensures fairness for long-lived background flows. Recall that our simulated network has 128 hosts, with bisection bandwidth of 1Gbps. We split the 128 hosts into 64 node-disjoint pairs. Between each pair we start N long-lived flows in both directions. If the network is stable, and DIBS does not induce unfairness, then we would expect each flow to get roughly 1/N Gpbs bandwidth, and the Jain's fairness index [11] would be close to 1. We carry out this experiment for N ranging from 1 to 16. Note that when N is 16, there 128 * 2 * 16 = 4096 long-lived TCP flows in the network. Our results show that Jain's fairness index is over 0.9 for all values of N.

5.7 When does DIBS break?

While DIBS performs well under a variety of traffic and network conditions, it important to note that beyond a level of congestion detouring packets (instead of dropping them) can actually hurt performance. However, for this to happen, the congestion has to be so extreme that the entire available buffer (across all switches) in the network gets used. In such a scenario, it is better to drop packets than to detour them, since there is no spare buffer capacity anywhere in the network. To understand where this breaking point is we push the workload, especially the query traffic, to extreme. The goal is to show that such a breaking point where DIBS can cause congestion to spread critically exists indeed, but for the tested scenario it is unrealistically high.

We start by pushing QPS to ever higher values. Figure 13 shows that, for the specific topology we tested against, DIBS breaks when we generate more than 10K queries per second. In this case, the queries arrive so fast that detoured packets do not get a chance to leave the network before new packets arrive. Thus, large queues build up in the entire network, which hurts performance of both query and background traffic.

Note that the size of the query response is small, and it takes the sender just two RTTs to send it to target receiver. Thus, DCTCP's congestion control mechanism (ECN marking) is not effective, since it requires several RTTs to work effectively.

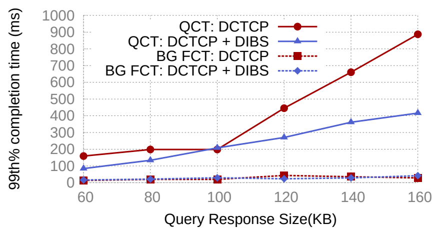
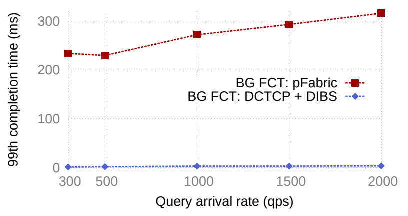
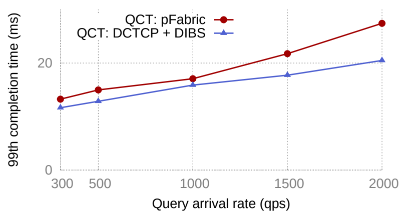


Figure 14: Large query response sizes (Background inter-arrival time: 120ms; incast degree: 40)



(a) Background traffic



(b) Query traffic

Figure 15: DIBS vs pFabric: Mixed traffic: Variable query arrival rate. (Background inter-arrival time: 120ms; incast degree: 40; response size: 20KB)

The exact tipping point depends on the specifics of the topology, the link speed, etc. Our goal is not to find it for all environments, but rather to show that such a tipping point exists, and that, for a reasonable setup, it is extremely high.

Next, we tried to increase query response sizes to see if we could obtain a similar tipping point. Query rate was held constant at 2000 queries per second. However, we found that DIBS does not "break" in this scenario. The reason is that the large query response size requires several RTTs to transmit, which gives DCTCP enough time to throttle the senders (Figure 14).

5.8 Comparison to pFabric

A number of recent proposals have shown that network performance can be significantly improved by assigning priorities to flows and taking these priorities into account when making scheduling decisions [20, 51]. The latest proposal in this area is pFabric [20], which provides near-optimal performance for high-priority traffic by maintaining switch queues by priority order, instead of FIFO.

We compare to pFabric as it is the state of the art, performing better than similar approaches like PDQ [34]. pFabric calls for very shallow buffers per port (24 packets) to minimize the overhead involved in maintaining a sorted packet queue. If a high-priority packet arrives at a queue that is full, a switch simply drops the lowest priority packet in the queue to make room for the new arrival. It relies on the end host to retransmit the dropped packet expeditiously. To this end, pFabric requires end hosts to run a modified version of TCP with a fixed timeout set to a very low value of 40 μs. However, the 40μs timeout that pFabric demands is difficult to implement in servers: most modern kernels have 1ms clock ticks. Additionally, packets need to be tagged with priorities, and switches must be able to drop or forward random packets in their queues selectively.

We now compare the performance between DIBS (i.e., DCTCP + DIBS) and pFabric. We use the same k=8 FatTree topology as earlier and we generate the same mixed workloads in both cases, as described in §5.3. For pfabric, the buffer size is set to 24 packets and the minRTO is adjusted to 350us since 1Gpbs links are used.

Figure 15(a) shows that pFabric can hurt performance of large background flows, since it gives higher priority to shorter flows. Thus, when query traffic (short flows) is high, long flows get starved. DIBS does not prioritize traffic, and hence does not suffer from this problem[8]. In fact, Figure 15(b) shows that at high query arrival rate, DIBS even slightly improves 99th percentile of the QCT for query traffic. This is because at high query arrival rates, pfabric drops too many packets, and ends up doing excessive retransmissions. DIBS reduces packet losses via detouring, and thus has slightly better performance.

5.9 Summary

In summary, these results show that DIBS performs well over a wide range of traffic. It improves the performance of query traffic while causing little or no collateral damage to the background flows. The results also show that under heavy load, DIBS is stable and long-lived flows treat each other fairly – although DIBS can lead to poor performance under extremely heavy query arrival rate.

6. Related Work

Data center networking is an active area of research. Here, we only discuss recent ideas, closely related to DIBS.

Hot potato routing: DIBS' main ancestor is hot potato (or deflection) routing [23] which established the basic premise to forward a packet to another node if it cannot be buffered while in transit. DIBS is the first to explore how this idea is uniquely suited for data center environments and workloads. There are several theoretical frameworks [26, 42, 45] designed to evaluate hot potato routing that can also be used as the basis for a theoretical analysis of DIBS.

Ethernet flow control: Hop-by-hop Ethernet flow control is designed for building lossless Ethernet networks[9]. When buffer of a switch gets full, it pauses its upstream switch,

[8] It is possible that not prioritizing short flows can hurt DIBS– e.g. if queue is already filled with packets from background flows right before an incast wave, then DIBS will detour packets from query flows (short), while pFabric will give them priority. However, these scenarios are less common and were not observed in our simulations.

[9] Infiniband [6] uses similar ideas, although it is a different L2 technology.

and the pause message eventually cascades to the sender. Priority flow control (PFC) [2, 31] expands on this basic idea to provide flow control for individual classes of service. PFC is leveraged by protocols like RoCE [13] and DeTail [53].

Like DIBS, Ethernet flow control may be viewed as a mechanism to implicitly allow switches to share buffers: when buffer usage at a switch exceeds a certain threshold, the pause message causes packets to queue up at the upstream switch.

However, DIBS does not guarantee a lossless network; it only minimizes losses in case of bursty traffic. Lossless L2 layer may be needed for specialized settings like high-performance compute clusters and storage area networks may need a lossless L2 layer. However, typical data center applications and the transport protocols (e.g. TCP) are designed to tolerate occasional packet loss.

Ethernet flow control can be difficult to tune. To avoid buffer overflow, a switch must send a pause message before its buffer actually fills up, since message propagation and processing takes time. Calculation of this threshold must account for cable lengths and switch architecture [1, 53]. To avoid buffer underflow, the pause duration must also be calculated carefully. In contrast, DIBS, with random detouring strategy, has no parameters, and thus requires no tuning.

DIBS also offers more flexibility than Ethernet flow control. With Ethernet flow control, buffer sharing happens only between a switch and its upstream neighbors. DIBS can redirect packets to any neighbor, including downstream ones.

DIBS is also free of problems such as deadlock [22], as we do not require any host or switch to stop transmitting.

Equal-cost multi-path (ECMP): ECMP spreads packets between a source and a destination across multiple routes, which is essential to data center topologies such as VL2 [32] and FatTree [16]. ECMP may be seen as a form of implicit buffer sharing among switches, since it splits traffic along multiple paths. However, ECMP and DIBS differ in several respects.

First, ECMP typically operates at flow level, while DIBS operates at packet level, achieving finer-grained buffer sharing at the expense of some packet reordering. While packet level ECMP has been proposed [30], it is not widely used. Second, DIBS spreads packets based on network load, not path length. While load aware ECMP has been proposed, it often requires complex centralized route management and hence is not practical. Third, ECMP, as the name implies, is limited to using equal cost paths. DIBS has no such restrictions. Most importantly, ECMP cannot provide succor in some traffic scenarios, such as incast. When multiple flows converge on a single receiver and the edge switch become a bottleneck, even packet-level, load-aware routing [28] will not help in this setting, while DIBS can.

Using ECMP does not rule out using DIBS. ECMP would do coarse-granularity load-spreading, while DIBS helps out on a shorter timescale. Indeed, in all the experiments shown in §5, we used DIBS with flow-level ECMP.

Multipath TCP (MPTCP): MPTCP [46] is a transport protocol that works with ECMP to ensure better load spreading. DIBS can co-exist with MPTCP.

Centralized traffic management systems: Centralized traffic management systems [17, 24, 50, 51, 53] collect traffic information in data centers and coordinate the hosts or switches to optimize flow scheduling. Thus, the centralized controller can only manage coarse-grained traffic at large timescale. DIBS can complement such systems, by mitigating packet losses arising from short-term behavior of flows that the centralized schemes cannot fully control.

Other transport protocols: In this paper, we coupled DIBS with DCTCP. It is possible to combine DIBS with other DCN transport protocols [52] as well, as long as requirements in (§3) are met.

Detouring in other settings The concept of detouring has been explored in scenarios as diverse as on-chip networks [43], optical networks [27], overlay networks [21, 25, 33, 47] and fast failure recovery schemes [39–41]. Like DIBS, flyways [37] rely on the fact that data center networks usually have sufficient capacity but experience localized congestion. Flyways provide spot relief using one-hop wireless detours.

7. Discussion

Network topology and detouring: In this paper, we focused on the FatTree topology. We now consider the effectiveness of DIBS in other topologies with different levels of path diversity. A switch gains more detouring options if it has more neighbors (higher degree), which are better for flow completion if they offer alternate paths to the destination. DIBS suffers if detouring options result in packets traversing long paths with lengthy queues, leading to timeouts or drops.[10]

Two recent topologies, HyperX [15] and JellyFish [48], seem to have properties well-suited for detouring. HyperX networks have many paths of different lengths between pairs of hosts. One can imagine using the short paths under normal conditions, but using detouring to exploit the larger path diversity when conditions warranted. Jellyfish connects fixed-degree switches together randomly to provide higher bandwidth than equivalently-sized FatTree topologies. To achieve these bandwidth gains, Jellyfish uses a fixed number of paths between pairs of hosts, some of which may be longer than the shortest ones. DIBS can detour packets to all these paths even they are of different length. Moreover, since Jellyfish

[10] To merely function correctly, DIBS does not need multiple disjoint paths between a sender and a receiver. In theory, DIBS would work even on a linear topology, where DIBS can either detour packets back on the reverse path, or, in the worst case, drop them.

has more switches that are closer to a destination than FatTree, DIBS to have more neighboring buffers to share.

Network admission control: Congestion mitigation is always coupled with network admission control. DCTCP controls the sending rate of long flows, thus admitting more short flows into the network. With DIBS, we admit even more short flows by sharing more buffers across switches. However, we still need admission control at the hosts to prevent applications from sending too many intensive short flows (e.g., due to misconfigurations, application bugs, or malicious users).

Other detouring policies: In this paper, we focus on simple random detouring without any parameter tuning. However, DIBS can provide highly flexible detouring policies by making different design decisions on (i) when to start detouring; (ii) which packets to detour; (iii) where to detour them to; and (iv) when to stop detouring. We discuss example detouring policies that may be useful in different settings, leaving detailed design and evaluation for future work.

Load-aware detouring: Random detouring works well in a FatTree topology, because ECMP is effective in splitting traffic equally among shortest paths. However, topologies such as Jellyfish and HyperX have paths with different lengths and have varying numbers of flows on these paths. Load-aware detouring detours packets to neighboring switches based on load. For example, when the destination port's buffer is full, a switch sends the packet via its output port that has the lowest current buffer usage.

Flow-based detouring: Our basic mechanism makes detouring decisions at the packet level, meaning that packets from the same flow can traverse different paths. Instead, switches could detour at the flow granularity, similar to how ECMP is usually deployed. Some flows would be detoured more often than others, and detoured packets from the same flow would follow a consistent path. An operator could even encode policy in the configurations for flow-based decisions in order to, for example, favor detouring of long flows, short flows, or flows from certain users.

Probabilistic detouring: Detouring can be used to provide different delays to different priorities of traffic. A switch can detour packets with different probabilities based on current buffer occupancy and packet priority. When the buffer is lightly loaded, the switch may only detour some of the lowest priority traffic to reserve room for higher-priority packets. As the buffer fills, the switch detours more classes of traffic with higher probability. By detouring different traffic with different probabilities, we essentially use a group of FIFO queues at different switches to approximate a priority queues at a single switch.

8. Conclusion

In this paper, we proposed detour-induced buffer sharing, which uses available buffer capacity in the network to handle sudden flashes of congestion. With DIBS, when a switch's buffer towards a destination is full, instead of dropping packets, it detours them to neighboring switches, achieving a near lossless network. In effect, DIBS provides a temporary virtual infinite buffer to deal with sudden congestion.

We implemented and evaluated DIBS using NetFPGA, Click and NS-3, under a wide range of conditions. Our evaluation shows that DIBS can handle bursty traffic, without interfering with their abilities to regulate routine traffic. This simple scheme is just the starting point for what we believe we can realize using more sophisticated detouring schemes.

References

[1] http://www.cisco.com/en/US/netsol/ns669/networking_solutions_solution_segment_home.html.

[2] 802.11p. http://en.wikipedia.org/wiki/IEEE_802.11p.

[3] Arista 7050qx-32. http://www.aristanetworks.com/media/system/pdf/Datasheets/7050QX-32_Datasheet.pdf.

[4] The emulab project. http://emulab.net/.

[5] Explicit congestion notification. http://tools.ietf.org/rfc/rfc3168.txt.

[6] Infiniband. http://en.wikipedia.org/wiki/Infiniband.

[7] iperf. http://iperf.fr.

[8] The netfpga project. http://netfpga.org/.

[9] The ns-3 project. http://www.nsnam.org.

[10] ns-3 simulator. www.nsnam.org.

[11] A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. In *ICNP*.

[12] Random early drop. http://tools.ietf.org/rfc/rfc2481.txt.

[13] Rdma over converged ethernet. http://en.wikipedia.org/wiki/RDMA_over_Converged_Ethernet.

[14] Tcp new reno. http://tools.ietf.org/html/rfc6582.

[15] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber. HyperX: Topology, routing, and packaging of efficient large-scale networks. In *SC*, 2009.

[16] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[17] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, pages 281–296, 2010.

[18] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[19] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is more: Trading a little bandwidth

for ultra-low latency in the data center. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, NSDI '12, San Jose, CA, USA, 2012. USENIX.

[20] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: minimal near-optimal datacenter transport. In *SIGCOMM*, pages 435–446, 2013.

[21] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *SOSP*, 2001.

[22] S. arne Reinemo and T. Skeie. Ethernet as a lossless deadlock free system area network. In *in Parallel and Distributed Processing and Applications: Third International Symposium, ISPA 2005*, pages 2–5. Springer Berlin/Heidelberg.

[23] P. Baran. On distributed communications networks. *Communications Systems, IEEE Transactions on*, 12(1):1–9, 1964.

[24] T. Benson, A. Anand, A. Akella, and M. Zhang. Microte: fine grained traffic engineering for data centers. In *CoNEXT*, page 8, 2011.

[25] C. Bornstein, T. Canfield, and G. Miller. Akarouting: A better way to go. In *MIT OpenCourseWare 18.996*, 2002.

[26] C. Busch, M. Herlihy, and R. Wattenhofer. Routing without flow control. In *SPAA*, pages 11–20, 2001.

[27] A. Busic, M. B. Mamoun, and J.-M. Fourneau. Modeling fiber delay loops in an all optical switch. In *international Conference on the Quantitative Evaluation of Systems*, 2006.

[28] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz. Per-packet load-balanced, low-latency routing for clos-based data center networks. CoNEXT '13. ACM, 2013.

[29] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding tcp incast throughput collapse in datacenter networks. In *WREN*, pages 73–82, 2009.

[30] A. Dixit, P. Prakash, and R. R. Kompella. On the efficacy of fine-grained traffic splitting protocols in data center networks. In *SIGCOMM poster*, 2011.

[31] O. Feuser and A. Wenzel. On the effects of the ieee 802.3x flow control in full-duplex ethernet lans. In *LCN*, pages 160–, 1999.

[32] A. G. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *SIGCOMM*, pages 51–62, 2009.

[33] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the reliability of Internet paths with one-hop source routing. In *OSDI*, 2004.

[34] C.-Y. Hong, M. Caesar, and B. Godfrey. Finishing flows quickly with preemptive scheduling. In *SIGCOMM*, pages 127–138, 2012.

[35] C. Hopps. Analysis of an equal-cost multi-path algorithm, 2000.

[36] S. Kandula and R. Mahajan. Sampling biases in network path measurements and what to do about it. In *IMC*, pages 156–169, 2009.

[37] S. Kandula, J. Padhye, and P. Bahl. Flyways to de-congest data center networks. In *HotNets*, 2009.

[38] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, Aug. 2000.

[39] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica. Achieving convergence-free routing using failure-carrying packets. In *SIGCOMM*, 2007.

[40] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker. Ensuring connectivity via data plane mechanisms. In *NSDI*, 2013.

[41] S. S. Lor, R. Landa, and M. Rio. Packet re-cycling: Eliminating packet losses due to network failures. In *HotNets*, 2010.

[42] N. F. Maxemchuk. Comparison of deflection and store-and-forward techniques in the manhattan street and shuffle-exchange networks. In *INFOCOM*, pages 800–809, 1989.

[43] T. Moscibroda and O. Mutlu. A case for bufferless routing in on-chip networks. In *ISCA*, 2009.

[44] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 39–50, New York, NY, USA, 2009. ACM.

[45] G. Nychis, C. Fallin, T. Moscibroda, O. Mutlu, and S. Seshan. On-chip networks from a networking perspective: congestion and scalability in many-core interconnects. In *SIGCOMM*, pages 407–418, 2012.

[46] C. Raiciu, S. Barré, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath tcp. In *SIGCOMM*, pages 266–277, 2011.

[47] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The end-to-end effects of Internet path selection. In *SIGCOMM*, 1999.

[48] A. Singla, C. yao Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking data centers, randomly.

[49] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller. Safe and effective fine-grained tcp retransmissions for datacenter communication. In *SIGCOMM*, pages 303–314, 2009.

[50] B. C. Vattikonda, G. Porter, A. Vahdat, and A. C. Snoeren. Practical tdma for datacenter ethernet. In *EuroSys*, pages 225–238, 2012.

[51] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: meeting deadlines in datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 50–61, New York, NY, USA, 2011. ACM.

[52] H. Wu, Z. Feng, C. Guo, and Y. Zhang. Ictcp: Incast congestion control for tcp in data center networks. In *CoNEXT*, page 13, 2010.

[53] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. H. Katz. Detail: reducing the flow completion time tail in datacenter networks. In *SIGCOMM*, pages 139–150, 2012.

[54] M. Zhang, B. Karp, S. Floyd, and L. L. Peterson. Rr-tcp: A reordering-robust tcp with dsack. In *ICNP*, pages 95–106, 2003.