DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks

David Zats∓, Tathagata Das∓, Prashanth Mohan∓, Dhruba Borthakur◇, Randy Katz∓
∓ University of California, Berkeley ◇ Facebook
{dzats, tdas, prmohan, randy}@cs.berkeley.edu, dhruba@fb.com

ABSTRACT

Web applications have now become so sophisticated that rendering a typical page may require hundreds of intra-datacenter flows. At the same time, web sites must meet strict page creation deadlines of 200-300ms to satisfy user demands for interactivity. Long-tailed flow completion times make it challenging for web sites to meet these constraints. They are forced to choose between rendering a subset of the complex page, or delay its rendering, thus missing deadlines and sacrificing either quality or responsiveness. Either option leads to potential financial loss.

In this paper, we present a new cross-layer network stack aimed at reducing the long tail of flow completion times. The approach exploits cross-layer information to reduce packet drops, prioritize latency-sensitive flows, and evenly distribute network load, effectively reducing the long tail of flow completion times. We evaluate our approach through NS-3 based simulation and Click-based implementation demonstrating our ability to consistently reduce the tail across a wide range of workloads. We often achieve reductions of over 50% in 99.9^{th} percentile flow completion times.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols

Keywords

Datacenter network, Flow statistics, Multi-path

1. INTRODUCTION

Web sites have grown complex in their quest to provide increasingly rich and dynamic content. A typical Facebook page consists of a timeline-organized "wall" that is writeable by the user and her friends, a real-time cascade of friend event notifications, a chat application listing friends currently on-line, and of course, advertisements selected by displayed content. Modern web pages such as these are made up of many components, generated by independent subsystems and "mixed" together to provide a rich presentation of information.

Building such systems is not easy. They exploit high-level parallelism to assemble the independent page parts in a timely fashion, and present these incrementally, subject to deadlines to provide an interactive response. The final mixing system must wait for all subsystems to deliver some of their content, potentially sacrificing responsiveness if a small number of subsystems are delayed. Alternatively, it must present what it has at the deadline, sacrificing page quality and wasting resources consumed in creating parts of a page that a user never sees.

In this paper, we investigate how the network complicates such application construction, because of the high variation in performance of the network flows underlying their distributed workflows. By improving the statistics of network flow completion, in particular *by reducing the long flow completion tail*, the application gains better worst-case performance from the network. Applying the end-to-end principle, while the mixer software must still deal with subsystems that fail to respond by the deadline, an underlying network that yields better flow statistics reduces the conservativeness of time-outs while reducing the frequency with which they are triggered. The ultimate application-layer result is better quality and responsiveness of the presented pages.

Deadlines are an essential constraint on how these systems are constructed. Experiments at Amazon [26] demonstrated that failing to provide a highly interactive web site leads to significant financial loss. Increasing page presentation times by as little as 100ms significantly reduces user satisfaction. To meet these demands, web sites seek to meet deadlines of 200-300ms 99.9% of the time [12, 33].

Highly variable flow completion times complicate the meeting of interactivity deadlines. Application workflows that span the network depend on the performance of the underlying network flows. Packet arrival pacing is dictated by round-trip-times (RTTs) and congestion can significantly affect performance. While datacenter network RTTs can be as low as $250\mu s$, in the presence of congestion, these times can grow by two orders of magnitude, forming a long tail distribution [12]. Average RTTs of hundreds of microseconds can occasionally take tens of milliseconds, with implications for how long a mixer application must wait before timing-out on receiving results from its subsystems.

Flash congestion is the culprit and it cannot be managed through conventional transport-layer means. Traffic bursts commonly cause packet losses and retransmissions [12]. Uneven load balancing often causes a subset of flows to experience unnecessarily high congestion [10]. The absence of traffic prioritization causes latency-sensitive foreground flows to wait behind latency-insensitive background flows [33]. Each contributes to increasing the long tail of flow completion*, especially for the latency-sensitive short flows critical for page creation*. While partial solutions exist [10, 12, 29,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCOMM'12, August 13–17, 2012, Helsinki, Finland.
Copyright 2012 ACM 978-1-4503-1419-0/12/08 ...$15.00.

33], no existing approach solves the whole problem. Fortunately, datacenter networks already contain the key enabling technology to reduce the long flow completion tail. They employ high-speed links and a scaled-out network topology, providing multiple paths between every source and destination [9, 23, 24].

Flash congestion can be reduced if it can be detected and if network-layer alternatives can be exploited quickly enough. We address this challenge by constructing a *cross-layer network stack that quickly detects congestion at lower network layers, to drive upper layer routing decisions, to find alternative lower-congestion paths to destinations.*

In this paper, we present the implementation and experimental evaluation of *DeTail*. DeTail is a cross-layer network stack design to reduce long-tailed flow completions in datacenter networks. It provides an effective network foundation for enabling mixer applications to assemble their complex content more completely and within responsiveness time constraints. The key contributions of this work are:

- Quantification of the impact of long-tailed flow completion times on different datacenter workflows;
- Assessment of the causes of long-tailed flow completion times;
- A cross-layer network stack that addresses them;
- Implementation-validated simulations demonstrating DeTail's reduction of 99.9^{th} percentile flow completion times by over 50% for many workloads without significantly increasing the median

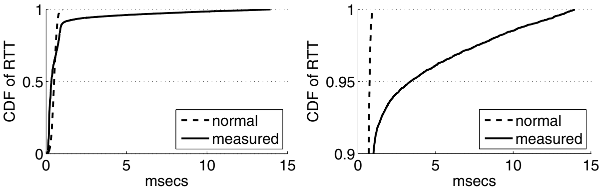
In the following section, we analyze how long-tailed flow completion times affect workflows' interactive deadlines. In Section 3, we describe the causes of long-tailed flow completion times and the inadequacy of partial solutions. In Section 4, we introduce the cross-layer network-based approach DeTail uses to overcome these issues. In Section 5, we describe the NS-3-based simulation [6] and Click-based implementation [27] with which we evaluate DeTail. The evaluation of DeTail in Section 6 demonstrates reduced flow completion times for a wide range of workloads. We discuss various aspects of DeTail in Section 7. We describe how DeTail compares with prior work in Section 8 and conclude in Section 9.

2. IMPACT OF THE LONG TAIL

In this section, we begin by analyzing datacenter network traffic measurements, describing the phenomenon of the long tail. Next, we present two workflows commonly used by page creation subsystems and quantify the impact of the long flow completion time tail on their ability to provide rich, interactive content. We compare this with the performance that could be achieved with shorter-tailed distributions. We conclude this section with a discussion of how to quantify the long tail.

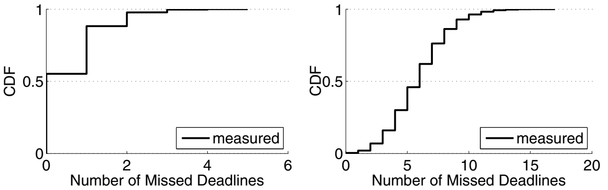
2.1 Traffic Measurements

Recently, Microsoft researchers [12] published datacenter traffic measurements for production networks performing services like web search. These traces captured three traffic types: (i) soft real-time queries, (ii) urgent short messages, and (iii) large deadline-insensitive background updates. Figure 1 reproduces graphs from [12], showing the distribution of measured round-trip-times (RTTs) from worker nodes to aggregators. The former typically communicate with mid-level aggregators (MLAs) located on the same rack. This graph represents the distribution of intra-rack RTTs.



(a) Complete Distribution (b) 90^{th}-100^{th} Percentile

Figure 1: CDF of RTTs from the worker to the aggregator. We compare Microsoft's measured distribution [12] with a synthetic normal one having a 50% larger median.



(a) 40 workers (b) 400 workers

Figure 2: Probability that a workflow will have a certain number of workers miss their $10ms$ deadline. All workers would meet their deadlines if RTTs followed the normal distribution.

Figure 1 shows that while the measured intra-rack RTTs are typically low, congestion causes them to vary by two orders of magnitude, forming a long-tail distribution. In this particular environment, intra-rack RTTs take as little as $61\mu s$ and have a median duration of $334\mu s$. But, in 10% of the cases, RTTs take over $1ms$. In fact, RTTs can be as high as $14ms$. These RTTs are the measured time between the transmission of a TCP packet and the receipt of its acknowledgement. Since switch queue size distributions match this behavior [11], *the variation in RTTs is caused primarily by congestion.*

For comparison, Figure 1 includes a synthetic distribution of RTTs following a normal distribution. While we set this distribution to have a median value that is 50% higher than that of the measured one, it has a much shorter tail.

As a measured distribution of datacenter flow completion times is unavailable, we conservatively assume each flow takes one RTT.

2.2 Impact on Workflows

Here we introduce the partition-aggregate and sequential workflows commonly used by page creation subsystems. For both workflows, we compare the impact of the long-tailed measured distribution with a shorter-tailed one. For this comparison, we focus on 99.9^{th} percentile performance as this is the common metric used for page creation [12, 33]. We see that a long-tailed distribution performs significantly worse than a shorter-tailed distribution, even when the latter has a higher median. We conclude this analysis with the key takeaways.

2.2.1 Partition-Aggregate

Partition-aggregate workflows are used by subsystems such as web search. Top-level aggregators (TLAs) receive requests. They divide (partition) the computation required to perform the request across multiple mid-level aggregators (MLAs), who further partition computation across worker nodes. Worker nodes perform the computation in parallel and send their results back to their MLA. Each MLA combines the results it receives and forwards them on to the TLA.

To ensure that the response is provided in a timely manner, it is common practice to give worker nodes as little as $10ms$ to perform their computation and deliver their result [12]. If a worker

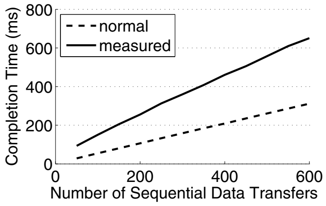


Figure 3: 99.9^{th} percentile completion times of sequential workflows. Web sites could use twice as many sequential requests per page under a shorter-tailed distribution.

node does not meet its deadline, its results are typically discarded, ultimately degrading the quality of the response.

To assess the impact of the measured RTT distribution (in Figure 1) on partition-aggregate workers meeting such deadlines, we analyze two hypothetical workflows. One has 40 workers while the other has 400. In Figure 2, we show the probability that a workflow will have a certain number of workers miss their deadlines. We assigned completion times to each worker by sampling from the measured RTT distribution. Those with completion times greater than $10ms$ were considered to have missed their deadlines. We performed this calculation 10000 times. Under the measured distribution, at the 99.9^{th} percentile, a 40-worker workflow has 4 workers (10%) miss their deadlines, while a 400-worker workflow has 14 (3.50%) miss theirs. Had RTTs followed the normal distribution, no workers would have missed their deadlines. This is despite the distribution having a 50% higher median than the measured one. This shows the hazard of designing for the median rather than long-tail performance.

These results assume that worker nodes do not spend any time computing the result they transmit. As the pressure for workers to perform more computation increases, the fraction of workers missing their deadlines will increase as well.

2.2.2 Sequential

In sequential workflows, a single front-end server fetches data from back-end servers (datastores) for every page creation. Future requests depend on the results of previous ones.

To quantify the impact of the long tail, we generated sequential workflows with varying numbers of data retrievals. We assumed that each data retrieval would use one flow and obtained values for retrievals by sampling from the appropriate distribution in Figure 1. We took the completion time of sequential workflows to be the sum of the randomly generated data retrieval times. We performed this calculation 10000 times.

In Figure 3, we report 99.9^{th} percentile completion times for different RTT distributions. Under the measured RTT distribution, to meet $200ms$ page creation deadlines, web sites must have less than 150 sequential data retrievals per page creation. Had RTTs followed the normal distribution, web sites could employ more than 350 sequential data retrievals per page. This is despite the distribution having a 50% higher median than the measured one. Again, designing for the median rather than long-tail performance is a mistake.

2.2.3 Takeaways

Long-tailed RTT distributions make it challenging for workflows used by page creation subsystems to meet interactivity deadlines. *While events at the long tail occur rarely, workflows use so many flows, that it is likely that several will experience long delays for every page creation.* Hitting the long tail is so significant that workflows actually perform better under distributions that have higher medians but shorter tails.

The impact is likely to be even greater than that presented here. Our analysis does not capture packet losses and retransmissions that are likely to cause more flows to hit the long tail.

Facebook engineers tell us that the long tail of flow completions forces their applications to choose between two poor options. They can set tight data retrieval timeouts for retrying requests. While this increases the likelihood that they will render complete pages, long tail flows generate non-productive requests that increase server load. Alternatively, they can use conservative timeouts that avoid unnecessary requests, but limit complete web page rendering by waiting too long for retrievals that never arrive. *A network that reduces the flow completion time tail allows such applications to use tighter timeouts to render more complete pages without increasing server load.*

2.3 Quantifying the Tail

Median flow completion time is an insufficient indicator of workflow performance. However, determining the right metric is challenging. Workflows only requiring 10 flows are much less likely to be affected by 99.9^{th} percentile flow completion times versus those with 1000 flows. To capture the effect of the long tail on a range of different workflow sizes, we report both 99^{th} and 99.9^{th} percentile flow completion times.

3. CAUSES OF LONG TAILS

Section 2 showed how the long tail of flow completion times impacts page creation workflows. As mentioned earlier, flash congestion aggravates three problems that lead to long-tailed flow completion times: packet losses and retransmissions, absence of prioritization, and uneven load balancing. Here we describe these problems and how they affect the latency-sensitive short flows critical to page creation. We then discuss why current solutions fall short.

3.1 Packet Losses and Retransmissions

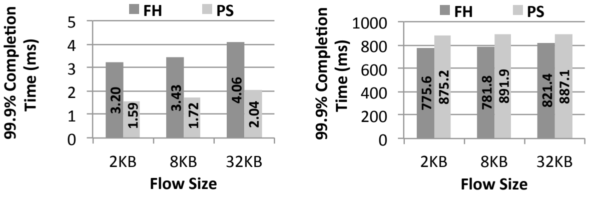
[12, 16, 31] study the effect of packet losses and retransmissions on network performance in datacenters. Packet losses often lead to flow timeouts, particularly in short flows where window sizes are not large enough to perform fast recovery. In datacenters, these timeouts are typically set to $10ms$ [12, 31]. Since datacenter RTTs are commonly of the order of $250\mu s$, just one timeout guarantees that the short flow will hit the long tail. It will complete too late, making it unusable for page creation. Using shorter timeouts may mitigate this problem, but it increases the likelihood of spurious retransmissions that increase network and server load.

Additionally, partition-aggregate workflows increase the likelihood of incast breakdown [12, 33]. Workers performing computation typically respond simultaneously to the same aggregator, sending it short flows. This sometimes leads to correlated losses that cause many flows to timeout and hit the long tail.

3.2 Absence of Prioritization

Datacenter networks represent a shared environment where many flows have different sizes and timeliness requirements. The traces from Section 2 show us that datacenters must support both latency-sensitive and latency-insensitive flows, with sizes that typically range from $2KB$ to $100MB$ [12].

During periods of flash congestion, short latency-sensitive flows can become enqueued behind long latency-insensitive flows. This increases the likelihood that latency-sensitive flows will hit the long tail and miss their deadlines. Approaches that do not consider different flow requirements can harm latency-sensitive flows.



(a) Regular Topology (b) Degraded Link

Figure 4: Simulated 99.9^{th} percentile flow completion times of flow hashing (FH) and packet scatter (PS)

3.3 Uneven Load Balancing

Modern datacenter networks have scaled out, creating many paths between every source and destination [9, 23, 24]. Flow hashing is typically used to spread load across these paths while maintaining the single-path assumption commonly employed by transport protocols. Imperfect hashing, as well as varying flow sizes often lead to uneven flow assignments. Some flows are unnecessarily assigned to a more congested path, despite the availability of less congested ones. This increases the likelihood that they will hit the long tail.

This phenomena has been observed before for large flow sizes [10, 29]. Here we show that it is also a problem for the short flows common in page creation. We present a simulation on a 128-server FatTree topology with a moderate oversubscription factor of four (two from top-of-rack to aggregate switches and two from aggregate to core switches). For this experiment, we ran an all-to-all workload consisting solely of high-priority, uniformly chosen $2KB$, $8KB$, and $32KB$ flows. These sizes span the range of latency-sensitive flows common in datacenter networks [12].

In Figure 4(a), we compare the performance of flow hashing and a simple multipath approach: *packet scatter*. Packet scatter randomly picks the output port on which to send packets when multiple shortest paths are available. To factor out transport-layer effects, we used infinitely large switch buffers and also disabled rate-limiting and packet retransmission mechanisms. We see that packet scatter significantly outperforms traditional flow hashing, cutting 99.9^{th} percentile flow completion times by half. As we have removed transport-layer effects, these results show that *single path approaches reliant of flow hashing significantly under-perform multipath ones.*

Multipath approaches that do not dynamically respond to congestion, like packet scatter, may perform significantly worse than flow hashing for topological asymmetries. Consider a common type of failure, where a $1Gbps$ link between a core and aggregate switch has been degraded and now operates at $100Mbps$ [29]. Figure 4(b) shows that for the same workload, packet scatter can perform 12% worse than flow hashing. As we will see in Section 6, flow hashing itself performs poorly.

Topological asymmetries occur for a variety of reasons. Datacenter network failures are common [18]. Asymmetries can be caused by incremental deployments or network reconfigurations. Both static approaches (packet scatter and flow hashing) are unaware of the different capabilities of different paths and cannot adjust to these environments. *An adaptive multipath approach would be able to manage such asymmetries.*

3.4 Current Solutions Insufficient

DCTCP, D^3, and HULL [12, 13, 33] are single path solutions recently proposed to reduce the completion times of latency-sensitive flows. Single-path fairness and congestion control protocols have also been developed through the datacenter bridging effort [2]. These reduce packet losses and prioritize latency-sensitive flows. But they do not address the uneven load balancing caused by flow hashing, and hence still suffer the performance loss illustrated in Figure 4(a).

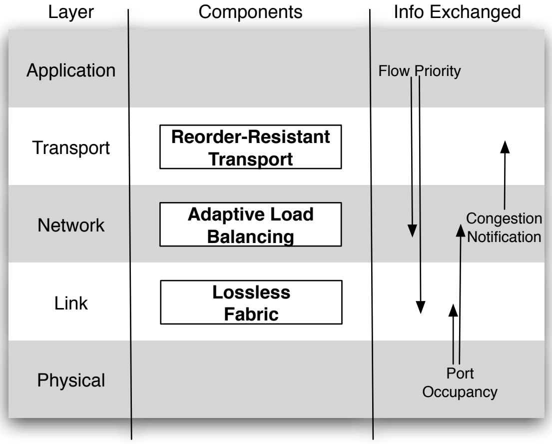


Figure 5: The DeTail network stack uses cross-layer information to address sources of long tails in flow completion times.

Recently two solutions have been proposed to more evenly balance flows across multiple paths. Hedera [10] monitors link state and periodically remaps flows to alleviate hotspots. Since Hedera remaps flows every five seconds and focuses on flows taking more than 10% of link capacity, it cannot improve performance for the short flows common in page creation.

The other solution is MPTCP [29]. MPTCP launches multiple TCP subflows and balances traffic across them based on congestion. MPTCP uses standard TCP congestion detection mechanisms that have been shown by DCTCP to be insufficient for preventing packet drops and retransmissions [12]. Also, while MPTCP is effective for flow sizes larger than $70KB$, it is worse than TCP for flows with less than 10 packets [29]. As small flows typically complete in just a few RTTs, host-based solutions do not have sufficient time to react to congested links and rebalance their load. *Current multipath-aware solutions cannot support the short flows common in page creation workflows.*

Most of the solutions discussed here seek to minimize in-network functionality. Instead they opt for host-based or controller-based approaches. Quick response times are needed to support the short, latency-sensitive flows common in page creation. In the following section, we present our network-oriented, cross-layer approach to meeting this goal.

4. DETAIL

In this section, we first provide an overview of DeTail's functionality and discuss how it addresses the causes of long-tailed flow completion times. We then describe the mechanisms DeTail uses to achieve this functionality and their parameterization.

4.1 Overview

DeTail is a cross-layer network-based approach for reducing the long flow completion time tail. Figure 5 depicts the components of the DeTail stack and the cross-layer information exchanged.

At the link layer, DeTail uses port buffer occupancies to construct a *lossless fabric* [2]. By responding quickly, lossless fabrics ensure that packets are never dropped due to flash congestion. They are only dropped due to hardware errors and/or failures. Preventing

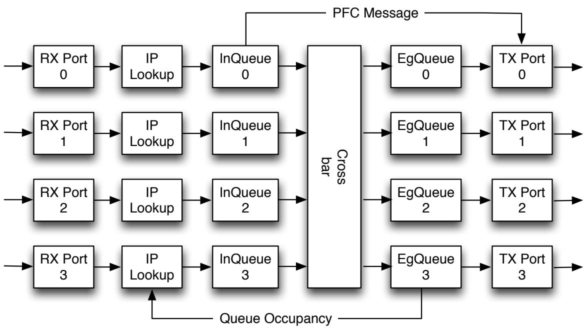


Figure 6: Assumed CIOQ Switch Architecture

congestion-related losses reduces the number of flows that experience long completion times.

At the network layer, DeTail performs per-packet adaptive load balancing of packet routes. At every hop, switches use the congestion information obtained from port buffer occupancies to dynamically pick a packet's next hop. This approach evenly smooths network load across available paths, reducing the likelihood of encountering a congested portion of the network. Since it is adaptive, it performs well even given topologic asymmetries.

DeTail's choices at the link and network layers have implications for transport. Since packets are no longer lost due to congestion, our transport protocol relies upon congestion notifications derived from port buffer occupancies. Since routes are load balanced one packet at a time, out-of-order packet delivery cannot be used as an early indication of congestion to the transport layer.

Finally, DeTail allows applications to specify flow priorities. Applications typically know which flows are latency-sensitive foreground flows and which are latency-insensitive background flows. By allowing applications to set these priorities, and responding to them at the link and network layers, DeTail ensures that high-priority packets do not get stuck behind low-priority ones. This assumes that applications are trusted, capable of specifying which flows are high priority. We believe that this assumption is appropriate for the kind of environment targeted by DeTail.

4.2 DeTail's Details

Now we discuss the detailed mechanisms DeTail uses to realize the functionality presented earlier. We begin by describing our assumed switch architecture. Then we go up the stack, discussing what DeTail does at every layer. We conclude by discussing the benefits of our cross-layer stack.

4.2.1 Assumed Switch Architecture

In Figure 6, we depict a four-port representation of a Combined Input/Output Queue (CIOQ) Switch. The CIOQ architecture is commonly used in today's switches [1,28]. We discuss DeTail's mechanisms in the context of this architecture and postpone discussion of others until Section 7. This architecture employs both ingress and egress queues, which we denote as InQueue and EgQueue, respectively. A crossbar moves packets between these queues.

When a packet arrives at an input port (e.g., RX Port 0), it is passed to the forwarding engine (IP Lookup). The forwarding engine determines on which output port (e.g., TX Port 2) the packet should be sent. Once the output port has been determined, the packet is stored in the ingress queue (i.e., InQueue 0) until the crossbar becomes available. When this happens, the packet is passed from the ingress queue to the egress queue corresponding to the desired output port (i.e., InQueue 0 to EgQueue 2). Finally, when the packet reaches the head of the egress queue, it is transmitted on the corresponding output port (i.e., TX Port 2).

To ensure that high-priority packets do not wait behind those with low-priority, the switch's ingress and egress queues perform strict priority queueing. Switches are typically capable of performing strict priority queueing between eight different priorities [4]. We use strict prioritization at both ingress and egress queues.

DeTail requires that the switch provide per-priority ingress and egress queue occupancies to higher layers in the stack. Each queue maintains a *drain bytes* counter per priority. This is the number of bytes of equal or higher priority in front of a newly arriving packet. The switch maintains this value by incrementing/decrementing the counters for each arriving/departing packet.

Having higher layers continuously poll the counter values of each queue may be prohibitively expensive. To address this issue, the switch associates a signal with each counter. Whenever the value of the counter is below a pre-defined threshold, the switch asserts the associated signal. These signals enable higher layers to quickly select queues without having to obtain the counter values from each. When multiple thresholds are used, a signal per threshold is associated with each counter. We describe how these thresholds are set in Section 4.3.2.

4.2.2 Link Layer

At the link layer, DeTail employs flow control to create a lossless fabric. While many variants of flow control exist [8], we chose to use the one that recently became part of the Ethernet standard: Priority Flow Control (PFC) [7]. PFC has already been adopted by vendors and is available on newer Ethernet switches [4].

The switch monitors ingress queue occupancy to detect congestion. When the drain byte counters of an ingress queue pass a threshold, the switch reacts by sending a Pause message informing the previous hop that it should stop transmitting packets with the specified priorities. When the drain byte counters reduce, it sends an Unpause message to the previous hop asking it to resume transmission of packets with the selected priorities[1].

During periods of persistent congestion, buffers at the previous hop fill, forcing it to generate its own Pause message. In this way, flow control messages can propagate back, quenching the source.

We chose to generate Pause/Unpause messages based on ingress queue occupancies because packets stored in these queues are attributed to the port on which they arrived. By sending Pause messages to the corresponding port when an ingress queue fills, DeTail ensures that the correct source postpones transmission.

Our choice of using PFC is based on the fact that packets in lossless fabrics can experience head-of-line blocking. With traditional flow control mechanisms, when the previous hop receives a Pause message, it must stop transmitting all packets on the link, not just those contributing to congestion. As a result, packets at the previous hop that are not contributing to congestion may be unnecessarily delayed. By allowing eight different priorities to be paused individually, PFC reduces the likelihood that low-priority packets will delay high priority ones. We describe how packet priorities are set in Section 4.2.5.

4.2.3 Network Layer

At the network layer, DeTail makes congestion-based load balancing decisions. Since datacenter networks have many paths between the source and destination, multiple shortest path options exist. When a packet arrives at a switch, it is forwarded on to the shortest path that is least congested.

---

[1] PFC messages specify the duration for which packet transmissions should be delayed. We use them here in an on/off fashion.

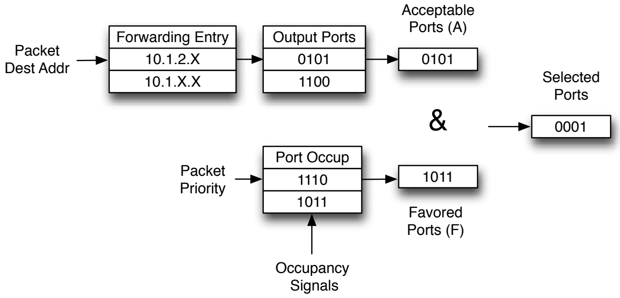


Figure 7: **Performing Adaptive Load Balancing** - A packet's destination IP address is used to determine the bitmap of *acceptable ports* (A). The packet's priority and port buffer occupancy signals are used to find the bitmap of the lightly loaded *favored ports* (F). A bitwise AND (&) of these two bitmaps gives the set of *selected ports* from which one is chosen.

DeTail monitors the egress queue occupancies described in Section 4.2.1 to make congestion-based decisions. Unlike traditional Ethernet, egress queue occupancies provide an indication of the congestion being experienced downstream. As congestion increases, flow control messages are propagated towards the source, causing the queues at each of the switches in the path to fill. By reacting to local egress queue occupancies we make globally-aware hop-by-hop decisions without additional control messages.

We would like to react by picking an acceptable port with the smallest drain byte counter at its egress queue for every packet. However, with the large number of ports in today's switches, the computational cost of doing so is prohibitively high. We leverage the threshold-based signals described in Section 4.2.1. By concatenating all the signals for a given priority, we obtain a bitmap of the favored ports, which are lightly loaded.

DeTail relies on forwarding engines to obtain the set of available shortest paths to a destination. We assume that associated with each forwarding entry is a bitmap of acceptable ports that lead to shortest paths for matching packets[2].

As shown in Figure 7, when a packet arrives, DeTail sends its destination IP address to the forwarding engine to determine which entry it belongs to and obtains the associated bitmap of acceptable ports (A). It then performs a bitwise *AND* (&) of this bitmap and the bitmap of favored ports (F) matching the packet's priority, to obtain the set of lightly loaded ports that the packet can use. DeTail randomly chooses from one of these ports and forwards the packet[3].

During periods of high congestion, the set of favored ports may be empty. In this case, DeTail performs the same operation with a second, larger threshold. If that does not yield results either, DeTail randomly picks a port from the bitmap. We describe how to set these thresholds in Section 4.3.2.

4.2.4 Transport Layer

A transport-layer protocol must address two issues to run on our load-balanced, lossless fabric. It must be resistant to packet reordering and it cannot depend on packet loss for congestion notification.

Our lossless fabric simplifies developing a transport protocol that is robust to out-of-order packet delivery. The lossless fabric ensures that packets will only be lost due to relatively infrequent hardware errors/failures. As packet drops are now much less frequent, it is not necessary that the transport protocol respond agilely to them. We simply need to disable the monitoring and reaction to out-of-order packet delivery. For TCP NewReno, we do this by disabling fast recovery and fast retransmit. While this leads to increased buffering at the end host, this is an acceptable tradeoff given the large amount of memory available on modern servers.

Obtaining congestion information from a lossless fabric is more difficult. Traditionally, transport protocols monitor packet drops to determine congestion information. As packet drops no longer happen due to congestion, we need another approach. To enable TCP NewReno to operate effectively with DeTail, we monitor the drain byte counters at all output queues. Low priority packets enqueued when the appropriate counter is above a threshold have their ECN flag set. This forces the low priority, deadline-insensitive TCP flow contributing to congestion to reduce its rate.

These types of modifications often raise concerns about performance and fairness across different transports. As the vast majority of datacenter flows are TCP [12] and operators can specify the transports used, we do not perform a cross-transport study here.

4.2.5 Application Layer

DeTail depends upon applications to properly specify flow priorities based on how latency-sensitive they are. Applications express these priorities to DeTail through the sockets interface. They set each flow (and hence the packets belonging to it) to have one of eight different priorities. As the priorities are relative, applications need not use all of them. In our evaluation, we only use two.

Applications must also react to extreme congestion events where the source has been quenched for a long time (Section 4.2.2). They need to determine how to reduce network load while minimally impacting the user.

4.2.6 Benefits of the Stack

DeTail's layers are designed to complement each other, overcoming limitations while preserving their advantages.

As mentioned earlier, link-layer flow control can cause head-of-line blocking. In addition to using priority, we mitigate this by employing adaptive load balancing and ECN. Adaptive load balancing allows alternate paths to be used when one is blocked and ECN handles the persistent congestion that aggravates head-of-line blocking.

DeTail's per-packet adaptive load balancing greatly benefits from the decisions made at the link and transport layers. Recall that using flow control at the link layer provides the adaptive load balancer with global congestion information, allowing it to make better decisions. And the transport layer's ability to handle out-of-order packet delivery allows the adaptive load balancer more flexibility in making decisions.

4.3 Choice of Settings

Now that we have described the mechanisms employed by DeTail, we discuss how to choose their parameters. We also assess how end-host parameters should be chosen when running DeTail.

4.3.1 Link Layer Flow Control

A key parameter is the threshold for triggering PFC messages. Pausing a link early allows congestion information to be propagated more quickly, making DeTail's adaptive load balancing more agile. At the same time, it increases the number of control messages. As PFC messages take time to be sent and responded to, setting the Unpause threshold too low can lead to buffer underflow, reducing link utilization.

To strike a balance between these competing concerns, we must first calculate the time to generate PFC messages. We use the same approach described in [7] to obtain this value.

---

[2] Bitmaps can be obtained with the TCAM and RAM approach as described in [9].

[3] Round-robin selection can be used if random selection is costly

For $1GigE$, it may take up to $36.456\mu s$ for a PFC message to take effect[4]. $4557B$ (bytes) may arrive after a switch generates a PFC message. As we pause every priority individually, this can happen for all eight priorities. We must leave $4557B \times 8 = 36456B$ of buffer space for receiving packets after PFC generation. Assuming 128*KB* buffers, this implies a maximum Pause threshold of $(131072B - 36456B)/8 = 11827$ *Drain Bytes* per priority. Setting the threshold any higher can lead to packet loss.

Calculating the Unpause threshold is challenging because the specifics of congestion cause queues to drain at different rates. Our calculations simply assume a drain rate of 1*Gbps*, requiring an Unpause threshold of at least $4557B$ to ensure the ingress queues do not overflow. However, ingress queues may drain faster or slower than $1Gbps$. If they drain slower, additional control messages may have to be sent, re-pausing the priority. If they drain faster, our egress queues reduce the likelihood of link underutilization.

These calculations establish the minimum and maximum threshold values to prevent packet loss and buffer underflow. Between the desire for agility and reduced control message overhead, we set the Unpause threshold to the minimum value of 4557 *Drain Bytes* and the Pause threshold to 8192 *Drain Bytes* (halfway between the minimum and the maximum). When fewer priorities are used, the Pause threshold can be raised without suffering packet loss. Given the desire for agile response to congestion, we leave it unmodified.

The tradeoffs discussed here depend on link speeds and buffer sizes. Analysis of how these tradeoffs change is left for future work.

4.3.2 Adaptive Load Balancing

When performing threshold-based adaptive-load balancing, we must determine how many thresholds to have for a given priority (i.e., most favored, favored, and least favored ports) as well as what these thresholds should be. Clearly, increasing the number of thresholds increases complexity, so the benefits of each additional threshold must outweigh the complexity cost.

Through a simulation-based exploration of the design space with the other parameters as described above, we determined that having two thresholds of 16*KB* and 64*KB* yields favorable results.

4.3.3 Explicit Congestion Notification

The threshold for setting ECN flags represents a tradeoff. Setting it too low reduces the likelihood of head-of-line blocking but increases the chance that low-priority flows will back off too much, underutilizing the link. Setting it too high has the opposite effect. Through experiments, we determined that a threshold of 64*KB* drain bytes appropriately makes this tradeoff.

4.3.4 End-Host Timers

Setting the timeout duration (i.e., RTO_{min} in TCP) of end host timers too low may lead to spurious retransmissions that waste network resources. Setting them too high leads to long response times when packets are dropped.

Traditionally, transport-layer protocols recover from packet drops caused by congestion and hardware failures. Congestion occurs frequently, so responding quickly to packet drops is important for achieving high throughput. However, DeTail ensures that packet drops only occur due to relatively infrequent hardware errors/failures. Therefore, it is more important for the timeout duration to be larger to avoid spurious retransmissions.

To determine a robust timeout duration for DeTail, we simulated all-to-all incast 25 times with varying numbers of servers (connected to a single switch) and different values of RTO_{min}. During every incast event, one server receives a total of 1*MB* from the remaining servers. We saw that values of $10ms$ and higher effectively avoid spurious retransmissions.

Unlike this simulation, datacenter topologies typically have multiple hops. Hence, we use $200ms$ as RTO_{min} for DeTail in our evaluations to accommodate the higher RTTs.

5. EXPERIMENTAL SETUP

Here we describe the NS-3 based simulator [6] and Click-based implementation [27] we use to evaluate DeTail.

5.1 NS-3 Simulation

Our NS-3 based simulation closely follows the switch design depicted in Figure 6. Datacenter switches typically have 128-256*KB* buffers per port [12]. To meet this constraint, we chose per-port ingress and egress queues of 128*KB*.

Network simulators typically assume that nodes are infinitely fast at processing packets, this is inadequate for evaluating DeTail. We extended NS-3 to include real-world processing delays. Switch delays of $25\mu s$ are common in datacenter networks [12]. We rely upon published specifications to break-down this delay as follows, providing explanations where possible:

- $12.24\mu s$ transmission delay of a full-size $1530B$ Ethernet frame on a 1*GigE* link.
- $3.06\mu s$ crossbar delay when using a speedup of 4. Crossbar speedups of 4 are commonly used to reduce head of line blocking [28].
- $0.476\mu s$ propagation delay on a copper link [7].
- $5\mu s$ transceiver delay (both ends of the link) [7].
- $4.224\mu s$ forwarding engine delay (the remainder of the $25\mu s$ budget).

We incorporate the transceiver delay into the propagation delay. The other delays are implemented individually, including the response time to PFC messages.

Packet-level simulators are known to have scalability issues, in terms of topology size and simulation duration [29]. We evaluated the feasibility of also developing a flow-level simulator, but concluded that it would be unable to shed light on the packet-level dynamics that are the focus of this paper.

NS-3's TCP model lacks support for ECN. Hence, our simulations do not evaluate explicit congestion notification (as discussed in Section 4.2.4). As we will show, even without ECN-based throttling of low priority flows our simulations demonstrate impressive results.

5.2 Click-based Implementation

To validate our approach, we implemented DeTail in Click [27]. Overall, our implementation mirrors the design decisions specified in Section 4 and portrayed in Figure 6. Here we describe the salient differences and analyze the impact they have on our parameters.

5.2.1 Design Differences

Unlike hardware switches, software routers typically do not emulate a CIOQ switch architecture. Instead, the forwarding engine places packets directly into the output queue. This output-queued approach is poorly suited to DeTail because we rely on ingress queues to determine when to send PFC messages.

---

[4] We do not consider jumbo frames. Also, PFC is only defined for 10*GigE*. We use 1*GigE* for manageable simulation times. We base PFC response times on the time specified for Pause Frames. This is appropriate since 10*GigE* links are given the same amount of time to respond to PFC messages are they are to Pause Frames.

To address this difference, we modified Click to have both ingress and egress queues. When packets arrive, the forwarding engine simply annotates them with the desired output port and places them in the ingress queue corresponding to the port on which they arrived. Crossbar elements then pull packets from the ingress queue to the appropriate egress queue. Finally, when the output port becomes free, it pulls packets from its egress queue.

Software routers also typically do not have direct control over the underlying hardware. For example, when Click *sends* a packet, it is actually enqueued in the driver's ring buffer. The packet is then DMAed to the NIC where it waits in another buffer until it is transmitted. In Linux, the driver's ring buffer alone can contain hundreds of packets. It is difficult for the software router to asses how congested the output link is when performing load balancing. Also, hundreds of packets may be transmitted between the time when the software router receives a PFC message and it takes effect.

To address this issue, we add rate limiters in Click before every output port. They clock out packets based on the link's bandwidth. This reduces packet buildup in the driver's and NIC's buffers, instead keeping those packets in Click's queues for a longer duration.

5.2.2 Parameter Modifications

The limitations of our software router impact our parameter choices. As it lacks hardware support for PFC messages, it takes more time both generate and respond to them.

Also, our rate limiter allows batching up to 6*KB* of data to ensure efficient DMA use. This may cause PFC messages to be enqueued for longer before they are placed on the wire and additional data may be transmitted before a PFC message takes effect. This also hurts high-priority packets. High priority packets will suffer additional delays if they arrive just after a batch of low priority packets has been passed to the driver.

To address these limitations, we increased our Pause / Unpause thresholds. However, instead of increasing ingress queue sizes, we opted to ensure that only two priorities were used at a time. This approach allows us to provide a better assessment of the advantages of DeTail in datacenter networks.

6. EXPERIMENTAL RESULTS

In this section, we evaluate DeTail through extensive simulation and implementation, demonstrating its ability to reduce the flow completion time tail for a wide range of workloads. We begin with an overview describing our traffic workloads and touch on key results. Next, we compare simulation and implementation results, validating our simulator. Later, we subject DeTail to a wide range of workloads under a larger topology than permitted by the implementation and investigate its scaled-up performance.

6.1 Overview

To evaluate DeTail's ability to reduce the flow completion time tail, we compare the following approaches:

Flow Hashing (*FH*): Switches employ flow-level hashing. This is the status quo and is our baseline for comparing the performance of DeTail.

Lossless Packet Scatter (*LPS*): Switches employ packet scatter (as already explained in Section 3) along with Priority Flow Control (PFC). While not industry standard, *LPS* is a naive multipath approach that can be deployed in current datacenters. The performance difference between *LPS* and DeTail highlights the advantages of Adaptive Load Balancing (ALB).

DeTail: As already explained in previous sections, switches employ PFC and ALB.

All three cases use strict priority queueing and use TCP NewReno as the transport layer protocol. For *FH*, we use a TCP RTO_{min} of $10ms$, as suggested by prior work [12, 31]. Since *LPS* and DeTail use PFC to avoid packet losses, we use the standard value of $200ms$ (as discussed in Section 4.3.4). Also, we use reorder buffers at the end-hosts to deal with out-of-order packet delivery.

We evaluate DeTail against *LPS* only in Section 6.4. For all other workloads, *LPS* shows similar improvements as DeTail and has been omitted for space constraints.

Traffic Model: Our traffic model consists primarily of high-priority data retrievals. For each retrieval, a server sends a 10-byte request to another server and obtains a variable sized response (i.e., data) from it. The size of the data (henceforth referred to as *retrieval data size*) is randomly chosen to be 2*KB*, 8*KB*, or 32*KB*, with equal probability. We chose discrete data sizes for more effective analysis of 99^{th} and 99.9^{th} percentile performance. The rate of generation of these data retrievals (henceforth called *retrieval rate*) and the selection of servers for the retrievals are defined by the traffic workload. In most cases, we assumed the inter-arrival times of retrievals to be exponentially distributed (that is, a Poisson process). We also evaluated against more bursty traffic models having lognormal distributions with varying sigma (σ) values. Where specified, we also run low-priority, long background data transfers.

Key results: Throughout our evaluation, we focus on 99^{th} and 99.9^{th} percentile completion times of data retrievals to assess DeTail's effectiveness. We use the percentage reduction in the completion times provided by DeTail over Flow Hashing as the metric of improvement. Our key results are:

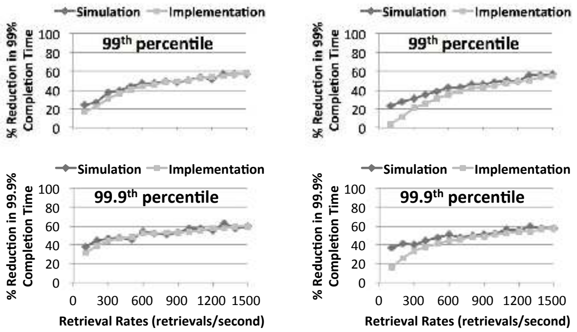
- DeTail completely avoids congestion-related losses, reducing 99.9^{th} percentile completion times of data retrievals in all-to-all workloads by up to 84% over Flow Hashing.
- DeTail effectively moves packets away from congestion hotspots that may arise due to disconnected links, reducing 99.9^{th} percentile completion times by up to 89% over Flow Hashing. *LPS* does not do as well and actually performs worse than *FH* for degraded links.
- Reductions in individual data retrievals translate into improvements for sequential and partition-aggregate workflows, reducing their 99.9^{th} percentile completion times by 54% and 78%, respectively.

6.2 Simulator Verification

To validate our simulator, we ran our Click-based implementation on Deter [14]. We constructed a 36-node, 16-server FatTree topology. Over-subscription is common in datacenter networks [3]. To model the effect of a moderate over-subscription factor of four, we rate-limited the ToR-to-aggregate links to 500*Mbps* and the aggregate-to-core links to 250*Mbps*.

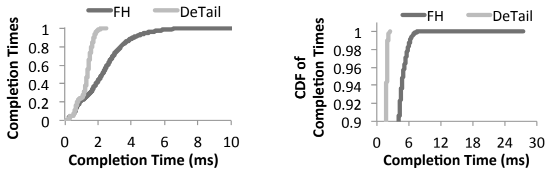
We designated half of the servers to be front-end (web-facing) servers and half to be back-end servers. Each front-end server continuously selects a back-end server and issues a high-priority data retrieval to it. The data retrievals are according to a Poisson process and their rate is varied from 100 to 1500 retrievals/second.

We simulated the same workload and topology, with parameters matched with that of the implementation. Figure 8 compares the simulation results with the implementation measurements. For rates ranging from 500 to 1500 retrievals/sec, the percentage reduction in completion time predicted by the simulator is closely matched by implementation measurements, with the difference in



(a) $2KB$ (b) $8KB$

Figure 8: **Comparison of simulation and implementation results** - Reduction by DeTail over *FH* in 99^{th} and 99.9^{th} percentile completion times of $2KB$ and $8KB$ data retrievals



(a) Complete distribution (b) 90^{th}-100^{th} percentile

Figure 9: CDF of completion times of $8KB$ data retrievals under all-to-all workload of 2000 retrievals/second

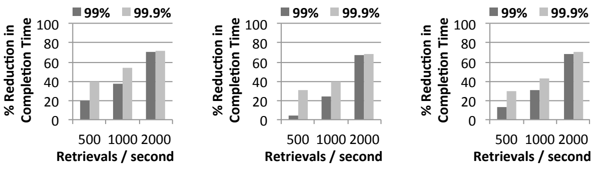
the percentage being within 8% (results for $32KB$ data retrievals and *LPS* are similar and have been omitted for space constraints). Note that this difference increases for lower rates. We hypothesize that this is due to end-host processing delays that are present only in the implementation (i.e., not captured by simulation) dominating completion times during light traffic loads.

We similarly verified our simulator for lognormal distributions of data retrievals having a $\sigma = 1$. The simulation and implementation results continue to match, with the difference in the percentage growing slightly to 12%.This demonstrates that our simulator is a good predictor of performance that one may expect in a real implementation. Next, we use this simulator to evaluate larger topologies and a wider range of workloads.

6.3 Microbenchmarks

We evaluate the performance of DeTail on a larger FatTree topology with 128 servers. The servers are distributed into four pods having four ToR switches and four aggregate switches each. The four pods are connected to eight core switches. This gives an oversubscription factor of four in the network (two from top-of-rack to aggregate switches and two from aggregate to core switches). We evaluate two traffic patterns:

- **All-to-all**: Each server randomly selects another server and retrieves data from it. All 128 servers engage in issuing and serving data retrievals.
- **Front-end / Back-end**: Each server in first three pods (i.e, front-end server) retrieves data from a randomly selected server in the fourth pod (i.e., back-end server).



(a) $2KB$ (b) $8KB$ (c) $32KB$

Figure 10: **All-to-all Workload** - Reduction by DeTail over *FH* in 99^{th} and 99.9^{th} percentile completion times of $2KB$, $8KB$ and $32KB$ retrievals

σ	0.5			1			2		
size (KB)	2	8	32	2	8	32	2	8	32
500 (r/s)	40%	20%	26%	38%	26%	26%	31%	30%	31%
1000 (r/s)	43%	30%	35%	46%	35%	37%	36%	23%	33%
2000 (r/s)	67%	62%	65%	68%	66%	67%	84%	76%	73%

Table 1: **All-to-all Workload with Lognormal Distributions** - Reduction in 99.9^{th} percentile completion time of retrievals under lognormal arrivals

The data retrievals follow a Poisson process unless mentioned otherwise. In addition, each server is engaged in, on average, one $1MB$ low-priority background flow. Using a wide range of workloads, we illustrate how ALB and PFC employed in DeTail reduce the tail of completion times as compared to *FH*.

All-to-all Workload: Each server generates retrievals at rates ranging from 500 to 2000 retrievals/second, which corresponds to load factors[5] of approximately 0.17 to 0.67. Figure 9 illustrates the effectiveness of DeTail in reducing the tail, by presenting the cumulative distribution of completion times of $8KB$ data retrievals under a rate of 2000 retrievals/second. While the 99^{th} and 99.9^{th} percentile completion times under *FH* were $6.3ms$ and $7.3ms$, respectively, DeTail reduced them to $2.1ms$ and $2.3ms$; a reduction of about 67% in both cases. Even the median completion time improved by about 40%, from $2.2ms$ to $1.3ms$. Furthermore, the worst case completion time was $28ms$ under *FH* compared to $2.6ms$, which demonstrates the phenomenon discussed in Section 2. Flow completion times can increase by an order of magnitude due to congestion and mechanisms employed by DeTail are essential for ensuring tighter bounds on network performance.

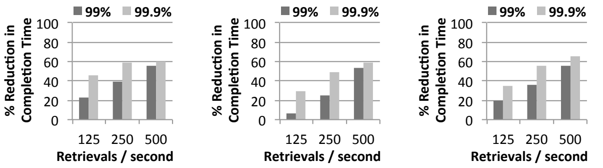
Figure 10 presents the reductions in completion times for three data sizes at three retrieval rates. DeTail provided up to 70% reduction at the 99^{th} percentile (71% at 99.9^{th} percentile) completion times. Specifically, the 99.9^{th} percentile completion times for all sizes were within $3.6ms$ compared to $11.9ms$ under *FH*. Within each data size, higher rates have greater improvement. The higher traffic load at these rates exacerbates the uneven load balancing caused by *FH*, which ALB addresses.

We also evaluate DeTail under more bursty traffic using lognormally distributed inter-arrival times. While keeping the same mean query rate (i.e., same load factors) as before, we vary the distribution's parameter σ from 0.5 to 2. Higher values of σ lead to more bursty traffic. Table 1 shows that DeTail achieves between 20% and 84% reductions at the 99.9^{th} percentile. Note that even at low load (500 r/s), for highly bursty ($\sigma = 2$) traffic DeTail achieves reductions greater than 30%.

Front-end / Back-end Workload: Each front-end server (i.e., servers in the first three pods) retrieves data from randomly selected back-end servers (i.e., servers in the fourth pod) at rates ranging from 125 to 500 retrievals/second, which correspond to load factors of approximately 0.17 to 0.67 on the aggregate-to-core links of the

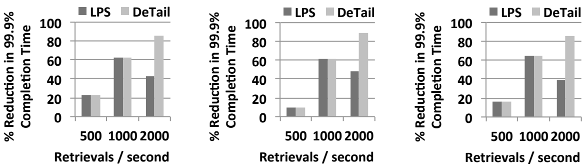
---

[5] load factor is the approximate utilization of the aggregate-to-core links by high-priority traffic only



(a) $2KB$ (b) $8KB$ (c) $32KB$

Figure 11: **Front-end / Back-end Workload** - Reduction by DeTail over *FH* in 99^{th} and 99.9^{th} percentile completion times of 2*KB*, 8*KB* and 32*KB* data retrievals



(a) $2KB$ (b) $8KB$ (c) $32KB$

Figure 12: **Disconnected Link** - Reduction by *LPS* and DeTail over *FH* in 99.9^{th} percentile completion times of 2*KB*, 8*KB* and 32*KB* retrievals

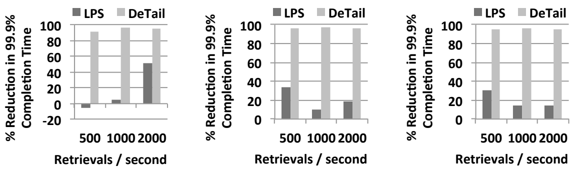
fourth pod. Figure 11 shows that DeTail achieves 30% to 65% reduction in the completion times of data retrievals at the 99.9^{th} percentile. This illustrates that DeTail can perform well even under the persistent hotspot caused by this workload.

Long Background Flows: DeTail's approach to improving data retrievals (i.e., high-priority, short flows) does not sacrifice background flow performance. Due to NS-3's lack of ECN support, we evaluate the performance of background flows using the 16-server implementation presented earlier. We use the same half front-end servers and half-backend servers setup, and apply a retrieval rate 300 retrievals/second. Additionally, front-end servers are also continuously engaged in low-priority background flows with randomly selected back-end servers. The background flows are long; each flow is randomly chosen to be one of 1*MB*, 16*MB* or 64*MB* with equal probability. Figure 14 shows that DeTail provides a 38% to 60% reduction over *FH* in the average completion time and a 58% to 71% reduction in the 99^{th} percentile. Thus, DeTail significantly improves the performance of long flows. A detailed evaluation of DeTail's impact on long flows is left for future work.

6.4 Topological Asymmetries

As discussed in Section 3.3, a multipath approach must be robust enough to handle topological asymmetries due to network component failures or reconfigurations. We consider two types of asymmetries: disconnected links and degraded links. These asymmetries lead to load imbalance, even with packet scatter. In this section, we show how ALB can adapt to the varying traffic demands and overcome the limitations of packet-level scattering. Besides *FH*, we evaluate DeTail against *LPS* to highlight the strength of ALB over packet scatter (used in *LPS*). We assume that the routing protocol used in the network has detected the asymmetry and converged to provide stable multiple routes.

Disconnected Link: We evaluated the all-to-all workload with Poisson data retrievals on the same topology described in the previous subsection, but with the assumption of one disconnected aggregate-to-core link. Figure 12 presents the reduction in 99.9^{th} percentile completion times for both *LPS* and DeTail (we do not present 99^{th} percentile for space constraints). DeTail provided 10% to 89% reduction, almost an order of magnitude improvement ($18ms$ under DeTail compared to $159ms$ under *FH* for 8*KB* retrievals at 2000 retrievals/second). *LPS's* inability to match DeTail's improvement at higher retrieval rates highlights the effectiveness of ALB at evenly distributing load despite asymmetries in available paths.



(a) $2KB$ (b) $8KB$ (c) $32KB$

Figure 13: **Degraded Link** - Reduction by *LPS* and DeTail over *FH* in 99.9^{th} percentile completion times of 2*KB*, 8*KB* and 32*KB* data retrievals

Degraded Link: Instead of disconnecting, links can occasionally be downgraded from 1*Gbps* to 100*Mbps*. Figure 13 presents the results for the same workload with a degraded core-to-agg link. DeTail provided more than 91% reduction compared to *FH*. This dramatic improvement is due to ALB's inherent capability to route around congestion hotspots (i.e., switches connected to the degraded link) by redirecting traffic to alternate paths. While the 99.9^{th} percentile completion time for 8*KB* at 2000 retrievals/second (refer to Figure 13(b)) under *FH* and *LPS* was more than $755ms$, it was $37ms$ under DeTail. In certain cases, *LPS* actually performs worse than *FH* (i.e., for 2*KB*, 500 retrievals/second).

In both fault types, the improvement in the tail comes at the cost of increased median completion times. As we have argued earlier, this trade off between median and 99.9^{th} percentile performance is appropriate for consistently meeting deadlines.

6.5 Web Workloads

Next, we evaluate how the improvements in individual data retrievals translate to improvements in the sequential and partition-aggregate workflows used in page creation. Here we randomly assign half the servers to be front-end servers and half to be back-end servers. The front-end servers initiate the workflows to retrieve data from randomly chosen back-end servers. We present the reduction in the 99.9^{th} percentile completion times of these workflows.

Sequential Workflows: Each sequential workflow initiated by a front-end server consists of 10 data retrievals of size 2*KB*, 4*KB*, 8*KB*, 16*KB*, and 32*KB* (randomly chosen with equal probability). As described in the Section 2, these retrievals need to be performed one after another. Workflows arrive according to a Poisson process at an average rate of 350 workflows/second. Figure 15 shows that DeTail provides 71% to 76% reduction in the 99.9^{th} percentile completion times of individual data retrievals. In total, there is a 54% improvement in the 99.9^{th} percentile completion time of the sequential workflows – from $38ms$ to $18ms$.

Partition-Aggregate Workflows: In each partition-aggregate workflow, a front-end server retrieves data in parallel from 10, 20, or 40 (randomly chosen with equal probability) back-end servers. As characterized in [12], the size of individual data retrievals is set to 2*KB*. These workflows arrive according to a Poisson process

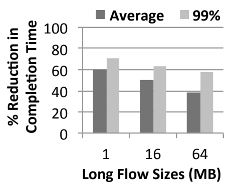


Figure 14: **Long Flows** - Reduction by DeTail in completion times of long, low-priority flows

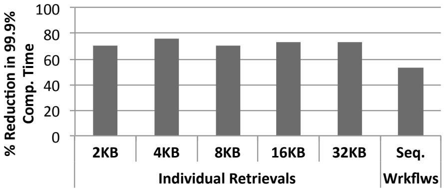


Figure 15: **Sequential Workflows** - Reduction by DeTail over *FH* in 99.9^{th} percentile completion times of sequential workflows and their individual data retrievals

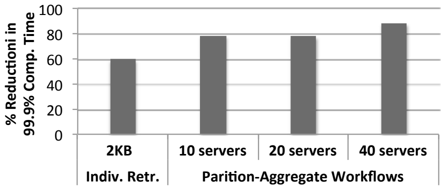


Figure 16: **Partition-Aggregate Workflows** - Reduction by DeTail over *FH* in 99.9^{th} percentile completion times of partition-aggregate workflows and their individual retrievals

at an average rate of 600 workflows/second. Figure 16 shows that Detail provides 78% to 88% reduction in 99.9^{th} percentile completion times of the workflows. Specifically, the 99.9^{th} percentile completion time of workflows with 40 servers was $17ms$ under DeTail, compared to $143ms$ under *FH*. This dramatic improvement is achieved by preventing the timeouts that were experienced by over 3% of the individual data retrievals under *FH*.

These results demonstrate that DeTail effectively manages network congestion, providing significant improvements in the performance of distributed page creation workflows.

7. DISCUSSION

We first describe how DeTail can be applied to other switch architectures. Next we present initial ideas about a DeTail-aware transport protocol.

7.1 Alternate Switch Architectures

Modern datacenters increasingly employ shared-memory top-of-rack switches [12]. In these switches, arriving packets are added to the output queue when the forwarding decision is made. They do not wait in input queues until the crossbar becomes available. This makes it difficult to determine which links contribute to congestion.

We address this by associating a bitmap with every input port. When an arriving packet is enqueued on a congested output queue, the bit corresponding to that port is set. When the output queue empties, the corresponding bits in the input ports are cleared. As input ports with any bits set in their bitmaps are contributing to congestion, this determines when we send Pause/Unpause messages. To handle multiple priorities, we use a per-port bitmap for each.

We have output queues only report congestion for a priority if its drain bytes have exceeded the thresholds specified earlier and if total queue occupancy is greater than 128*KB*. This reduces the likelihood of underflow in the same way that the 128*KB* output queues do in the CIOQ architecture (see Section 4).

To evaluate this approach, we re-ran the Poisson all-to-all microbenchmark presented in Section 6. As before, we assume our switches have 256*KB* per-port. Shared-memory architectures dynamically set queue occupancy thresholds. We simulated a simple model that optimizes per-port fairness. When a switch's memory is exhausted, it drops packets from the queue with the highest occupancy. Arriving packets may only be dropped if they are destined for the most occupied queue. Priority is used to decide which of an output queue's packets to drop. We believe this is an idealized model of the performance a shared-memory switch with the same optimization strategy can achieve.

In Table 2, we present the reduction in 99.9^{th} percentile data retrieval times. Due to space constraints, we do not present 99^{th} percentile results. With up to 66% reduction in completion time, these results show that DeTail's approach is beneficial for shared memory switches as well. We leave a thorough evaluation of DeTail's performance with shared-memory switches for future work.

	500			1000			2000		
size (KB)	2	8	32	2	8	32	2	8	32
reduction	17%	10%	14%	38%	34%	35%	66%	64%	66%

Table 2: **Shared Memory** - Reduction by DeTail in 99.9^{th} percentile completion times for all-to-all workloads of exponentially distributed retrievals

7.2 DeTail-aware transport

The transport layer protocol presented in this paper is a retrofit of TCP NewReno. Delay-based protocols, such as TCP Vegas [15], may be better suited in these environments. Instead of waiting for packet drops that do not occur, they monitor increases in delay. Increased delay is precisely the behavior our lossless interconnect exhibits as congestion rises. We plan to investigate this approach further in the future.

8. RELATED WORK

In this section, we discuss prior work and how it relates to DeTail in three areas: Internet protocols, datacenters, and HPC interconnects, discussing each in turn.

8.1 Internet Protocols

The Internet was initially designed as a series of independent layers [17] with a focus on placing functionality at the end-hosts [30]. This approach explicitly sacrificed performance for generality. Improvements to this design, in terms of TCP modifications such as NewReno, Vegas, and SACK [15, 20, 25] and in terms of buffer management such as RED and Fair Queuing [19, 21] were proposed. All of these approaches focused on improving the notification and response of end-hosts. Consequently, they operate at coarse-grained timescales inappropriate for our workload.

DeTail differs from this work by taking a more agile in-network approach that breaks the single path assumption to reduce the flow completion time tail.

8.2 Datacenter Networks

Relevant datacenter work has focused on two areas: topologies and traffic management protocols. Topologies such as FatTrees, VL2, BCube, and DCell [9, 22–24] sought to increase bisection bandwidth. Doing so necessitated increasing the number of paths between the source and destination because increasing link speeds was seen as impossible or prohibitively expensive.

Prior work has also focused on traffic management protocols for datacenters. DCTCP and HULL proposed mechanisms to improve flow completion time by reducing buffer occupancies [12, 13]. D^3 sought to allocate flow resources based on application-specified deadlines [33]. And, the recent industrial effort known as Datacenter Bridging extends Ethernet to support traffic from other protocols that have different link layer assumptions [2]. All of these approaches focus on single-path mechanisms that are bound by the performance of flow hashing.

Datacenter protocols focused on spreading load across multiple paths have been proposed. Hedera performs periodic flow remapping of elephant flows [10]. MPTCP takes a step further, making TCP aware of multiple paths [29]. While these approaches provide multipath support, they operate at timescales that are too coarse-grained to improve the short flow completion time tail.

8.3 HPC Interconnects

DeTail borrows some ideas from HPC interconnects. Credit-based flow control has been extensively studied and is often deployed to create lossless fabrics [8]. Adaptive load balancing algorithms such as UGAL and PAR have also been proposed [8]. To the best of our knowledge, these mechanisms have not been evaluated for web-facing datacenter networks focused on reducing the flow completion tail.

A commodity HPC interconnect, Infiniband, has made its way into datacenter networks [5]. While Infiniband provides a priority-aware lossless interconnect, it does not perform Adaptive Load Balancing (ALB). Without ALB, hotspots can occur, leading a subset of flows to hit the long tail. Host-based approaches to performing load-balancing, such as [32] have been proposed. But these approaches are limited because they are not sufficiently agile.

9. CONCLUSION

In this paper, we presented DeTail, an approach for reducing the tail of completion times of the short, latency-sensitive flows critical for page creation. DeTail employs cross-layer, in-network mechanisms to reduce packet losses and retransmissions, prioritize latency-sensitive flows, and evenly balance traffic across multiple paths. By making its flow completion statistics robust to congestion, DeTail can reduce 99.9^{th} percentile flow completion times by over 50% for many workloads.

DeTail's approach will likely achieve significant improvements in the tail of flow completion times for the foreseeable future. Increases in network bandwidth are unlikely to be sufficient. Buffers will drain faster, but they will also fill up more quickly, ultimately causing the packet losses and retransmissions that lead to long tails. Prioritization will continue to be important as background flows will likely remain the dominant fraction of traffic. And load imbalances due to topological asymmetries will continue to create hotspots. By addressing these issues, DeTail enables web sites to deliver richer content while still meeting interactivity deadlines.

10. ACKNOWLEDGEMENTS

This work is supported by MuSyC: "Multi-Scale Systems Center", MARCO, Award #2009-BT-2052 and AmpLab: "AMPLab: Scalable Hybrid Data Systems Integrating Algorithms, Machines and People", DARPA, Award #031362. We thank Ganesh Ananthanarayanan, David Culler, Jon Kuroda, Sylvia Ratnasamy, Scott Shenker, and our shepherd Jon Crowcroft for their insightful comments and suggestions. We also thank Mohammad Alizadeh and David Maltz for helping us understand the DCTCP workloads.

11. REFERENCES

[1] Cisco nexus 5000 series architecture. http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9670/white_paper_c11-462176.html.

[2] Data center bridging. http://www.cisco.com/en/US/solutions/collateral/ns340/ns517/ns224/ns783/at_a_glance_c45-460907.pdf.

[3] Datacenter networks are in my way. http://mvdirona.com/jrh/TalksAndPapers/JamesHamilton_CleanSlateCTO2009.pdf.

[4] Fulcrum focalpoint 6000 series. http://www.fulcrummicro.com/product_library/ FM6000_Product_Brief.pdf.

[5] Infiniband architecture specification release 1.2.1. http://infinibandta.org/.

[6] Ns3. http://www.nsnam.org/.

[7] Priority flow control: Build reliable layer 2 infrastructure. http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9670/white_paper_c11-542809.pdf.

[8] Abts, D., and Kim, J. High performance datacenter networks: Architectures, algorithms, and opportunities. *Synthesis Lectures on Computer Architecture 6*, 1 (2011).

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

[10] Al-fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. Hedera: Dynamic flow scheduling for data center networks. In *NSDI* (2010).

[11] Alizadeh, M. Personal communication, 2012.

[12] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data center tcp (dctcp). In *SIGCOMM* (2010).

[13] Alizadeh, M., Kabbani, A., Edsall, T., Prabhakar, B., Vahdat, A., and Yasuda, M. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *NSDI* (2012).

[14] Benzel, T., Braden, R., Kim, D., Neuman, C., Joseph, A., Sklower, K., Ostrenga, R., and Schwab, S. Experience with deter: a testbed for security research. In *TRIDENTCOM* (2006).

[15] Brakmo, L. S., O'Malley, S. W., and Peterson, L. L. Tcp vegas: new techniques for congestion detection and avoidance. In *SIGCOMM* (1994).

[16] Chen, Y., Griffith, R., Liu, J., Katz, R. H., and Joseph, A. D. Understanding tcp incast throughput collapse in datacenter networks. In *WREN* (2009).

[17] Clark, D. The design philosophy of the darpa internet protocols. In *SIGCOMM* (1988).

[18] Dean, J. Software engineering advice from building large-scale distributed systems. http://research.google.com/people/jeff/stanford-295-talk.pdf.

[19] Demers, A., Keshav, S., and Shenker, S. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM* (1989).

[20] Floyd, S., and Henderson, T. The newreno modification to tcp's fast recovery algorithm, 1999.

[21] Floyd, S., and Jacobson, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw. 1* (August 1993).

[22] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. Vl2: a scalable and flexible data center network. In *SIGCOMM* (2009).

[23] Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. Bcube: A high performance, server-centric network architecture for modular data centers. In *SIGCOMM* (2009).

[24] Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y., and Lu, S. Dcell: a scalable and fault-tolerant network structure for data centers. In *SIGCOMM* (2008).

[25] Jacobson, V., and Braden, R. T. Tcp extensions for long-delay paths, 1988.

[26] Kohavi, R., and Longbotham, R. Online experiments: Lessons learned, September 2007. http://exp-platform.com/Documents/IEEEComputer2007 OnlineExperiments.pdf.

[27] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. The click modular router. *ACM Trans. Comput. Syst. 18* (August 2000).

[28] McKeown, N. White paper: A fast switched backplane for a gigabit switched router. http://www-2.cs.cmu.edu/ srini/15-744/readings/McK97.pdf.

[29] Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., and Handley, M. Improving datacenter performance and robustness with multipath tcp. In *SIGCOMM* (2011).

[30] Saltzer, J. H., Reed, D. P., and Clark, D. D. End-to-end arguments in system design. *ACM Trans. Comput. Syst. 2* (November 1984).

[31] Vasudevan, V., Phanishayee, A., Shah, H., Krevat, E., Andersen, D. G., Ganger, G. R., Gibson, G. A., and Mueller, B. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *SIGCOMM* (2009).

[32] Vishnu, A., Koop, M., Moody, A., Mamidala, A. R., Narravula, S., and Panda, D. K. Hot-spot avoidance with multi-pathing over infiniband: An mpi perspective. In *CCGRID* (2007).

[33] Wilson, C., Ballani, H., Karagiannis, T., and Rowtron, A. Better never than late: meeting deadlines in datacenter networks. In *SIGCOMM* (2011).