DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks

David Zats
Tathagata Das
Prashanth Mohan
Randy H. Katz

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2011-113
http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-113.html

October 19, 2011

DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks

David Zats *UC Berkeley*
Tathagata Das *UC Berkeley*
Prashanth Mohan *UC Berkeley*
Randy Katz *UC Berkeley*

Abstract

Web sites are increasingly backed by complex processing to deliver rich content to users via web pages. Despite the increased complexity, the pages must still be delivered quickly and consistently to meet user's expectations for interactivity. To achieve this, datacenters typically employ application-level mechanisms to squeeze in as much complex processing as possible while still meeting page delivery deadlines. However, network variability can result in variable packet latency and a long flow completion time tail. This ultimately leads to either reduced page quality to meet deadlines or increased deadline misses.

In this paper, we evaluate the benefits of a new network congestion management approach for reducing the flow completion time tail. We argue that in-network traffic management, multipath data transfers, and traffic differentiation are essential for reducing the tail. We validate our approach through DeTail, an in-network multipath-aware, congestion management mechanism that reduces the flow completion tail, increasing the likelihood that complex web sites will be able to meet interactive deadlines. We show that DeTail effectively reduces the 99th percentile flow completion tail for a wide range of steady and bursty workloads by up to 80%.

1 Introduction

Web sites have grown increasingly complex in the quest to create richer content. Consider a typical Facebook page. It consists of notifications of user events, a chat application, as well as advertisements. Every web page is made up of many components, generated by independent subsystems and integrated together to provide a rich user experience.

In addition to increased web page complexity and richness, users continue to demand good interactive responsiveness. Experiments have demonstrated that failing to provide a highly interactive web site can lead to significant financial loss [23]. Even increasing page creation times by 100 ms can negatively impact user satisfaction. To address these issues, today's web sites often focus on meeting strict page creation deadlines of 200-300 ms 99.9% of the time [12, 34].

Construction of a web page from more primitive components may require complex processing and communication across many servers in the datacenter. Components such as web search typically leverage partition-aggregate workflows, dividing queries across many worker nodes and aggregating the results [12]. On the other hand, web pages consisting of many simpler components depend on front-end servers to perform data requests to many back-end servers for every page creation [28]. Network flows facilitate these types of inter-server communication.

Highly variable flow completion times complicate the meeting of interactivity deadlines. Network flows are made up of packets whose pacing is dictated by round-trip delays. Even though datacenter network round-trip-times can be as low as $250\mu s$, in the presence of congestion, they can grow by two orders of magnitude forming a long tail distribution [12]. Round-trip-times that average hundreds of microseconds can occasionally take tens of milliseconds. This variability in packet delivery yields flow completion times with a long tail, which may either cause degraded web pages to be constructed or deadlines to be missed.

Managing congestion sufficiently well to achieve predictable flow completion times is challenging. Modern datacenter topologies scale out, leveraging multiple links to increase bisection bandwidth [10, 19]. These topologies are characterized by large fan-in and fan-out points. Congestion grows rapidly at fan-in points when multiple incoming links transmit to the same outgoing link. Poorly spreading traffic at fan-out points increases the likelihood of congestion hotspots. Additionally, datacenters typically exhibit a mixture of flows with different completion deadlines [34]. Buffer management approaches that do not differentiate between these flows can harm those that are deadline-sensitive. All of these issues must be addressed to effectively reduce the flow completion time tail.

In this paper, we propose DeTail, an in-network, multipath aware, congestion management mechanism. To the best of our knowledge, DeTail is the first to focus on reducing the flow completion time tail in datacenter networks. It achieves this through a synthesis of a small set of judiciously chosen mechanisms that effectively work together to reduce network variability. DeTail employs the following mechanisms to effectively manage congestion:

- in-network traffic management to quickly detect and respond to rapid congestion increases;
- multipath data transfers to spread traffic across alternate paths, helping to avoid congestion hotspots;

- traffic differentiation to allocate network resources based on flow requirements.

Our contribution in this paper is to demonstrate how these mechanisms work together in a cohesive manner to reduce the flow completion time tail in datacenters.

To demonstrate the implementation feasibility of DeTail, we created a DeTail-compliant switch design. Our design focuses on reducing costs by leveraging on mechanisms commonly available in Ethernet switches wherever possible (i.e., priority flow control).

We use NS-3[6] simulations to extensively analyze and evaluate how DeTail reduces the flow completion time tail across a wide range of synthetic workloads. We begin with a range of steady and bursty workloads to assess its envelope of performance, demonstrating that DeTail consistently improves 99th percentile flow completion times by up to 80%. Next, we verify that DeTail's improvements are maintained in more complex workloads seen in web-facing datacenter environments. Finally, we evaluate our Click-based [24] implementation of DeTail to demonstrate its practicality.

In the following section, we analyze the effect of long-tailed flow completion times on the ability of datacenter applications to meet interactive deadlines. In Section 3, we detail the network requirements of a reduced tail solution for datacenter networks. In Section 4, we discuss our design of DeTail. Section 5 presents our switch-level implementation and Section 6 examines how the parameters should be chosen. In Section 7, we describe the methodology we used to evaluate DeTail. We present our evaluation results in Section 8. We discuss related work in Section 9 and conclude in Section 10.

2 Effect of Long-Tailed Flow Completions

Modern web sites must respond quickly to user actions. To meet this requirement, web sites strive to ensure that it take no longer than 200-300ms to create web pages [12, 34]. Failure to meet these deadlines can lead to user dissatisfaction, and therefore financial loss. For example, a 100ms increase in page load times decreased Amazon's sales by 1% [23]. In this section, we argue that variability in flow completion times is a significant cause that makes meeting deadlines hard.

It is critical that page creation deadlines be met consistently. Typically 99.9th percentile performance is used to evaluate consistency [12, 34]. At this percentile, up to 1 in a 1000 page creations may miss their interactive deadline. Given the scale of these web sites, using a lower percentile would result in a prohibitively large number of users experiencing slow or degraded pages. Hence it is essential that these deadlines be met at least 99.9% of the time.

At the same time, web sites increasingly employ complex workflows to provide a richer user experience. In this paper, we consider two particular cases - (i) partition-aggregate workflows and (ii) sequential workflows. Both of these workflows leverage many servers in the datacenter to perform complex operations while meeting page creation deadlines.

In partition-aggregate workflows, an aggregator node divides (partitions) computation across multiple worker nodes. Worker nodes perform the computation in parallel and send their results back to the aggregator. The aggregator combines these results to provide a complete response. To ensure that the complete response is obtained within the deadline, worker nodes may be given as little as 10 ms to perform their computation and deliver their result. If worker nodes do not meet their deadline, their results are typically discarded [12]. Increasing the amount of time that worker nodes can take to perform these computations is important. Mechanisms such as DCTCP have been developed to reduce flow completion times, enabling worker nodes to perform longer computations.

Other types of web sites employ sequential workflows where a single front-end server fetches data from back-end datastores for every page creation. For example, a page load at Facebook consists of an average of 130 data requests [28]. These requests often have sequential dependencies, i.e. future requests depend on the results of previous ones. Obtaining the responses of hundreds of sequential requests within hundreds of milliseconds is a significant challenge.

For workflows to meet these strict deadlines, requests and responses, and hence the underlying network flows, must complete on time. Depending on the connection state and the window size, it may take multiple round-trip-times (RTTs) to deliver a request or response. As demonstrated in [12], while RTTs can be as low as $250\mu s$, *congestion can cause queuing delays, increasing RTTs by two orders of magnitude*. Furthermore, these measurements indicate that the distribution of RTTs has a long tail, with 10% taking over 1 ms. In fact, just one RTT may potentially take 14 ms, causing requests and responses to miss their deadlines. As a result, this variability forces workflows to choose between reducing the quality of the result and missing deadlines.

Today, this problem is mitigated through application-level approaches that limit the richness of web pages. In partition-aggregate workflows, application-level jittering is employed to reduce the pathological network effects that cause high flow completion times [12]. On the other hand, web sites employing sequential workflows overcome network issues by aggressively reducing the number of data retrievals and using denormalized data [28].

In DeTail, we attack the root cause of this problem: long-tailed flow completion times due to poor congestion management. To improve the ability of web sites

to meet their deadlines, we focus on ensuring that 99th percentile flow completion times are reduced irrespective of load. Our choice of 99th percentile flow completion time reflects the fact that each web page consists of many flows that effectively represent sample points. It is well known that as more sample points are averaged, the distribution becomes tighter. So, by ensuring that 99% of flows complete quickly, we increase the likelihood that web pages consistently meet their deadlines.

In the next section, we discuss the requirements a datacenter network must meet to effectively reduce the flow completion time tail.

3 Datacenter Network Congestion Management Requirements

As described in the previous section, better congestion management is required to reduce the flow completion time tail. In this section, we begin by providing an overview of datacenter network attributes that make managing congestion challenging. Next, we delve into the requirements datacenter network mechanisms must meet to effectively manage congestion.

3.1 Datacenter network attributes

Datacenter networks typically have two properties that should be considered when designing a congestion management mechanism. The topologies they use and the traffic mix they support both have implications.

Modern datacenter topologies have scaled out, employing multiple links to increase bisection bandwidth [10, 19]. As a result, these topologies are typically characterized by large fan-in and fan-out points. At fan-in points switches may have to contend with the congestion caused by many incoming links simultaneously sending data to the same outgoing link. At fan-out points, switches have to decide on which of the outgoing links to send incoming data. As we will show later in this section, both fan-in and fan-out points need to be considered to effectively manage congestion.

Datacenters are typically characterized by a shared environment with many applications vying for resources [31]. Hence, datacenter networks have to support a traffic mix consisting of flows with various sizes and deadlines [34]. Flow sizes can range from 2KB - 100 MB and are expected to support application deadlines as short as 10 ms [12].

In the rest of this section, we will describe why in-network traffic management is effective at managing fan-in points and why multipath data transfers are useful for managing fan-out points. We also explain the utility of traffic differentiation.

3.2 In-network traffic management

Most protocols maintain the Internet's decision to place congestion control and reliability mechanisms at the end hosts [12, 34, 30]. This decision places a fundamental limit on the agility of the network in reacting to congestion. It takes approximately one RTT for (i) the sender to be notified of congestion and (ii) the impact of the sender's response to be felt at the bottleneck link. Thus, even an optimal host-based mechanism cannot resolve congestion in less than one RTT.

A response time of one RTT is too slow for delivering consistent flow completion times to datacenter applications. At a fan-in point, it can take less than one RTT for the outgoing link's queues to build-up and even overflow, leading to timeouts and unpredictable flow completion times. Furthermore, the same congestion that causes queues to build up also increases RTT variabilty, making end host congestion control even less responsive.

Additionally, protocols often measure network state indirectly. For example, TCP uses estimation to determine the presence of congestion. These protocols cannot react until an easily measurable event (i.e., a packet drop) has occurred. These events may only occur well after buffers have been exhausted. At this point, flows are already taking an unacceptably long time to complete. Attempts to resolve this problem by learning optimal window sizes over long periods of time are well suited in environments where long-lived connections are common [12]. However, they are not sufficiently general to effectively support workloads consisting of connections that are too short for the optimal window size to be learned.

In-network traffic management mechanisms can effectively manage the rapid increases in congestion that can occur in datacenter networks. Additionally, they can make better-informed decisions based on directly measured network state.

3.3 Multipath data transfer

Most transport protocols were designed with the assumption that only a single path would be used by a flow between a source and a destination [34, 12, 17]. These approaches do not effectively leverage the multiple paths available between every source and destination in datacenter networks.

The mechanisms typically used to spread traffic load while maintaining the single path assumption [10, 11] are insufficient [29]. Forwarding decisions based on flow hashing spread traffic unequally across links because they do not consider varying flow sizes. Centralized approaches mitigate this problem, but they do not operate at the frequency necessary to meet our performance requirements.

Mechanisms that provide in-network support for multipath data transfers address this issue. They enable traf-

fic to be spread more evenly on a per-packet basis instead of a per-flow basis. Additionally, by operating inside the network, they allow decisions to be quickly made at fan-out points instead of waiting for the host to respond.

3.4 Traffic differentiation

As mentioned earlier, congestion spikes are likely to occur at fan-in points. In-network traffic management and multipath data transfers reduce the impact of congestion spikes on the network as a whole. However, as they do not consider the requirements of individual flows, deadline-sensitive flows can still be unnecessarily delayed.

Traffic differentiation mechanisms are an effective approach for addressing this issue. By considering flow requirements when allocating bandwidth and buffers, these mechanisms ensure that resources will go to the flows that need them most.

4 Design of DeTail

In the previous section, we motivated the need to use in-network traffic management, multipath data transfers, and traffic differentiation to effectively manage congestion in datacenter networks. In this section, we describe the mechanisms DeTail employs to meet each of these requirements. We conclude this section with a discussion of how DeTail incorporates these mechanisms to perform in-network, multipath aware congestion management.

4.1 In-network traffic management

In DeTail, we employ link-layer-flow-control (LLFC) to enable quick, in-network responses based on direct measurement of congestion. By employing LLFC, switches learn from and react to congestion much more quickly than host-based approaches. Switches directly measure buffer occupancy and notify upstream switches when packet transmissions should be delayed. This approach completely avoids congestion-related packet losses in the network, eliminating timeouts and improving the consistency of flow completion times.

4.2 Multipath data transfer

We employ per-packet adaptive load balancing (ALB) to effectively spread traffic across available paths. When a packet arrives, our switch forwards it to the valid port with the smallest output queue. Through this approach, we reduce the occurrence of hot spots that lead to unpredictable flow completion times.

It needs to be noted that the use of multiple paths to transfer the data in each flow increases the likelihood of out-of-order packet delivery. Transport protocols like TCP may interpret packet reordering as losses, leading to unnecessary retransmissions. Since we already use in-network traffic management to prevent congestion-related losses, we can remedy this situation by employing a simple re-order buffer at the end-hosts.

4.3 Traffic differentiation

DeTail leverages priority mechanisms to improve the completion times of deadline-sensitive flows in the presence of congestion spikes. By employing priority mechanisms, we provide additional resources to deadline-sensitive flows at the expense of deadline-insensitive flows. Link-level priority mechanisms such as priority queueing have been studied extensively and are available in today's switches [4]. Additionally, our switches consider priority when performing load balancing and flow control. By leveraging priority, we ensure a greater consistency in the flow completion times for deadline-sensitive flows in datacenter environments.

4.4 Discussion

DeTail incorporates link-layer flow control, adaptive load balancing, and priority to perform in-network, multpath aware congestion management. As we will detail in the rest of this section, we chose these mechanisms because they operate in a cohesive manner, overcoming the limitations of using each individually.

While link-layer flow control mechanisms eliminate congestion-related packet drops, they are sometimes avoided because of the possibility of head-of-line blocking [7]. This occurs when the switch asks the previous hop to postpone packet transmissions over a certain link. All of packets scheduled for transmission on that link are delayed even if they are not contributing to congestion. DeTail reduces the impact of this phenomenon by employing per-packet adaptive load balancing. Through this approach, packets that would otherwise be delayed are transmitted on alternate links.

Per-packet adaptive load balancing techniques evenly spread load across available links. However, they are often avoided because of the challenges that packet reordering places on host-based transport protocols. DeTail addresses this issue by employing link-layer flow control to guard against congestion-related packet drops. With packet drops now only occurring due to hardware failures or bit errors, it is less challenging for end hosts to handle out-of-order packet delivery while maintaining high performance. As a result, adaptive load balancing mechanisms gain the flexibility to reorder packets.

Both link-layer flow control and adaptive load balancing techniques must effectively handle the highly-variable traffic mix that is common in datacenter networks. By leveraging priority, DeTail ensures that the most time-sensitive flows in this traffic mix are effectively handled.

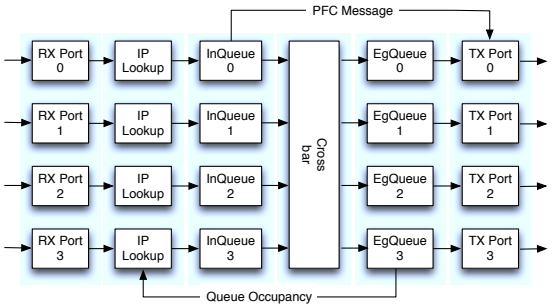


Figure 1: DeTail-compliant Switch Design

5 Switch-level implementation of DeTail

In this section, we describe our switch-level implementation of DeTail. We begin by providing an overview of the underlying switch architecture. Subsequently, we detail how the components of DeTail (link-layer flow control, adaptive load balancing, and prioritization) operate in the context of this switch architecture. We conclude this section with a discussion of how these components leverage each other to improve performance and touch upon their feasibility for datacenter networks.

In the remainder of this paper, we will evaluate our approach through both simulation and a software router implementation.

5.1 Switch Architecture

In Figure 1, we depict the four-port representation of a DeTail-compatible switch. As indicated in this figure, we use the standard Combined Input Output Queue (CIOQ) architecture that is commonly used in today's switches [26]. This architecture employs both ingress and egress queues, which we denote as InQueue and EgQueue, respectively. Additionally, this architecture utilizes a crossbar, which we schedule using the iSlip algorithm [27].

When a packet arrives at an input port (say, RX Port 0), it is passed to the forwarding engine (IP Lookup). The forwarding engine determines on which output port (say, 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 the crossbar becomes available, 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).

5.2 Link Layer Flow Control

DeTail employs LLFC to guard against congestion-related losses in datacenter networks. Many variants of flow control mechanisms exist, with credit-based flow control being commonly used in HPC interconnects [9]. To reduce the cost of implementing DeTail, we chose to leverage a flow control mechanism that is already part of the Ethernet standard - Pause Frames [8].

When a switch starts to experience congestion, it sends a Pause Frame to the previous hop. The Pause Frame informs the recipient that it should postpone packet transmissions for a specified duration. Once the pause frame expires, the recipient is expected to resume transmission as before.

Ingress queues represent a natural location for placing the logic for generating Pause Frames. Packets stored in these queues can be trivially attributed to the port on which they arrived. By sending Pause Frames to the corresponding port when an ingress queue fills up, DeTail ensures that the correct source postpones transmission. Additionally, as ingress queues are typically located next to the corresponding port, generating and sending a Pause Frame does not involve crossbar communication.

Analogously, egress queues and output ports are the logical place to respond to Pause Frames. By placing the logic at this location, we ensure that no packet will be transmitted on the link for the specified duration. To provide complete functionality, we also require logic in the egress queues that prevents buffer overflows. This logic ensures that packets will not be dropped in egress queues in periods of congestion and will instead be enqueued in the ingress queues. When these ingress queues fill up, they will generate Pause Frames, forcing previous hops to postpone transmission in the same manner. In this way, congestion notifications that cannot be resolved locally will propagate from the bottleneck link all the way back to the source.

5.3 Adaptive Load Balancing

In DeTail, we employ adaptive load balancing to spread traffic across available links, reducing the likelihood of congestion hot spots. To perform ALB, we extend the logic of the egress queues and the forwarding engines (IP Lookup).

We require our egress queues to maintain a counter specifying the number of bytes currently enqueued. Associated with each of these counters is a signal. Whenever the value of the counter is below a pre-defined threshold, a logical 1 is asserted on the associated signal. Otherwise, a zero is asserted. By concatenating all these signals, we obtain a bitmap of the favored ports, (F), which are lightly loaded.

Forwarding engines typically employ content addressable memory (TCAM) to store forwarding entries, speeding up longest prefix matching. As shown in prior work [10] and depicted in Figure 2, TCAMs can be used to reference a RAM entry containing additional information. In our case, these RAM entries store a bitmap of acceptable ports (A) on which matching packets can be transmitted.

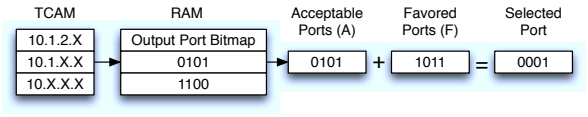


Figure 2: Using TCAM to perform Adaptive Load Balancing

When a packet arrives, DeTail sends its destination IP address to the TCAM to determine which entry it belongs to and obtains the associated bitmap of acceptable ports (A) from RAM. DeTail then performs a bitwise *AND* (&) of this bitmap (A) and the set of favored ports (F) to obtain the set of lightly loaded ports that the packet can use. DeTail randomly chooses from one of these remaining ports and forwards the packet.

During periods of high congestion, the set of favored ports may be empty. In this case, DeTail will randomly pick a port from the bitmap obtained by the TCAM lookup (A).

5.4 Priority

DeTail considers the priority of each flow when performing Link Layer Flow Control (LLFC), queueing packets in ingress and egress queues, and performing Adaptive Load Balancing (ALB).

To enable priority-based LLFC decisions, we use an extension to Pause Frames, known as Priority Flow Control (PFC) [7]. PFC is a recently standardized protocol that allows eight different priorities to be paused individually. PFC has already been adopted by vendors and is available on newer Ethernet switches [4].

PFC was designed as one of the datacenter bridging protocols. Consequently, the eight different priority levels were intended to enable different protocols with different L2 reliability assumptions (i.e., Fiber Channel) to share Ethernet links. In this work, we use PFC to differentiate among different types of TCP flows. Networks that require both types of differentiation may require an extension to PFC that provides more priority levels.

DeTail also employs priority mechanisms in ALB. The intuition is that with priority queues, simply considering the queue occupancy of the egress queues is insufficient. Take, for example, the case where a packet with priority 7 can be forwarded to output port 1 or 2. Output port 1 has a queue of 10 KB with priority 7. However, port 2 has a queue of 20 KB with priority 0. Based only on queue occupancy, ALB would send the packet to output port 1. However, if it considers priority and sends the packet to output port 2, the packet will be placed on the wire much sooner.

To efficiently consider multiple priorities, each egress queue maintains a counter per priority. Each counter records the number of *drain bytes* for its priority. The number of *drain bytes* is the number of bytes that must be transmitted before a new packet with the given priority can be placed on the wire. Since we employ strict priority queueing, the number of *drain bytes* can simply be computed as the number of enqueued bytes with higher or equal priority. This value can be maintained by simply incrementing/decrementing the counters for each arriving/departing packet.

Given these counters, DeTail constructs a bitmap of favored ports, (F) for each priority. When a packet arrives, its priority is determined and the appropriate bitmap of favored ports is selected. This bitmap is used in the same way described earlier to determine which output port to forward the packet to.

5.5 Discussion

In the remainder of this section, we discuss how the components of DeTail depend on each other to meet our performance goals. We also analyze the feasibility of our approach for datacenter networks.

5.5.1 Component Interdependence

The constituent components of DeTail depend on each other to meet our performance requirements. Furthermore, the performance of each one greatly improves when the others are present.

As discussed in the previous section, link-layer flow control (LLFC) mechanisms typically suffer from head-of-line blocking. Not only does adaptive load balancing (ALB) ameliorate this problem, but priority helps as well. Priority ensures that high priority packets will not suffer when low-priority ones cause congestion.

LLFC also assists ALB beyond reducing the need for hosts to respond aggressively to out of order packets. As mentioned earlier, priority flow control PFC effectively propagates congestion information from the bottleneck link back to the sources. A congested switch will send PFC messages to previous hop switches, asking them to postpone packet transmissions. The switches receiving PFC messages will enqueue packets, instead of sending them. As their egress and then ingress queues fill up, they will send their own PFC messages to their previous hops. Consequently the egress queue based decisions performed by ALB incorporate the congestion information provided by PFC.

Finally, priority mechanisms are only sufficient on their own when there are just a few high-priority flows. Whenever there are large numbers of high priority flows, high priority packets will still get lost due to buffer overflows. Thus, our priority mechanisms depend on ALB and LLFC to prevent these queue buildups.

5.5.2 Feasibility Analysis

DeTail depends on the ability of datacenter networks to employ custom switches. This assumption is appropriate for modern-day datacenters because operators are showing a willingness to adopt new designs [34].

Today, building custom switches is less of a hurdle than it was in the past. Custom switches can be built from commodity ASICs. These ASICs are reasonably priced; a PCI-E board with a switch ASIC costs as little as $400 [25]. Furthermore, ASIC manufacturers are showing a willingness to adopt new protocols. For example, the FM 6000 ASIC already incorporates the recently standardized datacenter bridging protocols [4].

However, we recognize that it is advantageous to leverage existing mechanisms, where possible, to reduce implementation cost. Our choice of using PFC to perform link-layer flow control was based on this consideration.

6 DeTail Parameters

In the previous section, we discussed how our Link Layer Flow Control (LLFC) and Adaptive Load Balancing (ALB) mechanisms employ thresholds when determining the action to take. In this section, we provide analysis of how to choose these parameters. We conclude this section with an assessment of how end host parameters should be chosen when running DeTail.

6.1 Link Layer Flow Control

The LLFC thresholds must satisfy multiple requirements that make choosing them challenging. Switches must transmit PFC messages early enough so that they can absorb the burst of packets that will arrive before the messages take effect. Additionally, switches must carefully consider how long to pause the link for, else they risk underflowing queues and wasting resources.

To reduce decision-making complexity, switches typically use PFC in an on/off fashion [7]. Instead of calculating how long to pause the link for, they send PFC messages with the max duration. When queue occupancy drops below a certain threshold, they unpause the link by sending another PFC message with the duration set to 0. We take the same approach in DeTail.

As described in [7], to calculate these thresholds we must determine how long it takes for a generated PFC message to take effect. When a switch generates a PFC message, it is enqueued for transmission at the head of the queue. Since the ongoing packet transmission may have just started, the message must wait T_O time for this transmission to complete. Next the PFC message encounters a transmission and propagation delay T_P as it travels from the sending switch to the recipient. The recipient is then given T_R time to react to the message. An ongoing packet transmission may have started just before the recipient reacts, in effect delaying the response for T_O time. Even, after the recipient has reacted, there may still be T_P worth of bytes traveling in the wire.

Taking all of these delays into account, the amount of time it takes to respond to a PFC message is given by:

$$T = T_O + T_P + T_R + T_O + T_P \quad (1)$$

If we consider Gigabit Ethernet links, the worst-case T_O is $1530B/1Gbps = 12.24\mu s$ [1]. Assuming copper links and (conservative) transmitter delays of $2.5\mu s$ [7], our maximum propagation delay is $T_P = 6.6\mu s$. According to the Ethernet standard, the switch has two 512-bit-times to react, or $1.024\mu s$ [2]. In total, it may take up to $38.7\mu s$ to react to a PFC message.

Assuming 1 Gbps links, 4838 bytes may arrive after a switch generates a PFC message. Furthermore, since we pause every priority individually, this can happen for all eight priorities. Thus, we must leave $4,838B \times 8 = 38,704B$ of buffer space for receiving packets after PFC generation. Assuming 128 KB buffers, this implies a high threshold of $(131,072B - 38,704B)/8 = 11,546B$ drain bytes per priority.

Now we must determine the thresholds for unpausing priorities. Setting an appropriate threshold is important because we want to ensure that the queues do not underflow. In our calculations, we assume that enqueued packets will drain at 1 Gbps. Therefore, we must unpause priorities when their drain byte counters drop below $4,838B$ to start receiving new packets before the ingress queue drains completely. In certain situations, ingress queues may drain faster or slower than 1 Gbps. If they drain slower, additional PFC messages may have to be sent, repausing the priority. If they drain faster, then the egress queues reduce the likelihood link underutilization.

6.2 Adaptive Load Balancing

Ideally, when performing Adaptive Load Balancing (ALB), the algorithm would pick the egress queue with the smallest number of drain bytes for the given priority. Since this may be prohibitively expensive, we use the threshold approach described earlier.

Given our approach, 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 the thresholds should be. Clearly increasing the number of thresholds increases complexity so the benefits of each additional threshold must outweigh the complexity cost.

Through our simulations we determined that having two thresholds of 16 KB and 64 KB yields favorable results. However satisfactory results can also be obtained for switches that can only support one threshold per priority.

6.3 End-Host Timers

End-host timers represent a tradeoff. Setting them too low leads to spurious retransmissions that waste network resources. Setting them too high leads to long response-times when packets are dropped.

TCP uses end-host timeouts to detect packet drops that are both due to congestion and hardware failures. Congestion occurs frequently, so responding quickly

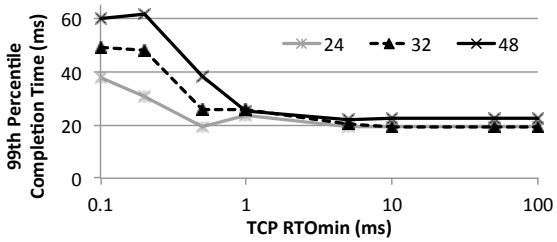


Figure 3: All-to-all Incast with varying numbers of servers connected to the same switch. RTOs less than 10 ms cause spurious retransmissions.

to packet drops is important for achieving high performance. Conversely, DeTail only experiences packet drops due to relatively infrequent hardware failures. Therefore, it is more important for the timeouts to be larger to avoid spurious retransmissions.

To determine the optimal timeout value for DeTail, we simulated all-to-all Incast for 25 iterations with varying numbers of servers and timeout values. During every incast, one server receives a total of 1 MB from the remaining servers. We plot the 99th percentile completion times for this experiment in Figure 3. As shown from this figure, timeouts 10 ms and larger are optimal for avoiding spurious retransmissions.

In this simulation, there is only one switch between all of the servers. So for the rest of our simulations we use a timeout of 50 ms to reflect that datacenters topologies typically have multiple hops.

7 Methodology

In this section, we describe the methodology used to evaluate DeTail. We begin by delving into the switch model used for our NS-3-based simulations. We use this model to evaluate DeTail's performance across a wide range of workloads, demonstrating the generality of our approach. Afterwards, we describe the Click-based implementation.

7.1 Simulation Model

Our NS-3 based simulation closely follows the switch design presented in Figure 1. Datacenter switches typically have 128 - 256 KB buffers per port [12]. Given this constraint as well as the need to absorb packets after sending PFC messages, we chose per-port ingress and egress queues of 128 KB.

Network simulators typically assume that nodes are infinitely fast at processing packets. Maintaining this assumption would not allow us to effectively evaluate DeTail. So, we extended NS-3 to include real-world processing delays. Switch delays of $25\mu s$ are common in datacenter networks [12]. We 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 [26].
- $1.6\mu s$ propagation delay on a copper link [7].
- $5\mu s$ transceiver delay (both ends of the link) [7].
- $3.1\mu s$ forwarding engine delay (the remainder of the $25\mu s$ budget).

In our simulation, we incorporate the transceiver delay into the propagation delay. All of the other delays are implemented individually, including the response time to PFC messages.

7.2 Implementation

To validate our approach, we also implemented DeTail in Click [24]. Click is a software router platform that allows the user to express functionality by composing elements. In general, our implementation mirrors the design decisions specified in Section 4 and depicted in Figure 1. In this section, we describe the salient differences and analyze the impact they have on our parameters.

7.2.1 Design Differences

Unlike hardware switches, software routers typically do not have a notion of a crossbar. Instead, packets are typically placed directly into the output queue by the forwarding engine. This output-queued approach is poorly suited to DeTail because we rely on ingress queues to determine when to send PFC messages.

To address this issue, 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. Whenever the output port becomes free, it pulls packets from the egress queue. This action causes the egress queue to pull multiple packets from the ingress queues in the order specified by iSlip [27].

Software routers also typically do not have direct control over the underlying hardware resources. 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 can be placed on the wire. In Linux, the driver's ring buffer alone can contain hundreds of packets. Thus it is difficult for the software router to asses how congested the output link is when performing load balancing. Worse yet, hundreds of packets may be transmitted between the time when the software router receives a PFC message and it takes effect.

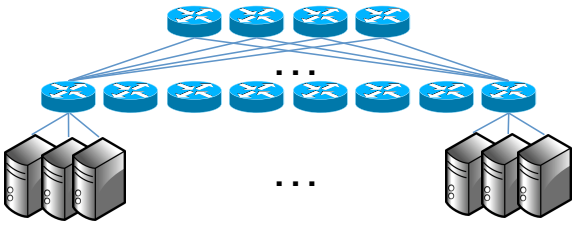


Figure 4: Simulation topology with 8 racks of 12 servers each

To address this issue, we add a rate limiter in Click right before every output port. Our rate limiters clock out packets based on the bandwidth of the link. As a result, they effectively reduce packet buildup in the driver's and NIC's buffers, instead keeping those packets in Click's queues for a longer duration. To ensure that packet buildup does not occur in the driver or the NIC, the rate limiters clock out packet at a rate that is 2% slower than line rate. While effective, this approach negligibly impacts performance.

7.2.2 Parameter Modifications

The design differences of our software router impact our parameter choices. Our software router does not have hardware support for PFC messages. Consequently it takes more time for them to be generated and responded to. Additionally, our rate limiter allows for 6 KB of outstanding data to ensure efficient DMA use. As a result, PFC messages may be enqueued for $48\mu s$ before they are placed on the wire and an additional 6 KB of data may be transmitted before a PFC message takes effect.

Addressing these physical constraints requires making decisions about pause/unpuase thresholds and ingress queue sizes. To absorb worst-care traffic bursts, we were forced to make a tradeoff between two options: increasing the buffer size or ensuring that only two priorities are used at a time. Given our desire to provide a clear assessment of the advantages of DeTail in datacenter networks, we opted for the reduced priority approach.

8 Evaluation

In this section, we evaluate DeTail, both through simulation and implementation.

8.1 Simulation

In our simulations, we used the topology shown in Figure 4. The simulation has 8 racks with 12 servers each. These racks are interconnected by a multi-rooted tree with an oversubscription factor of 3, which is moderate compared to the ratios typically used in today's datacenters [3].

We begin by putting DeTail through a series of basic traffic workloads, and contrasting its performance with the individual constituent components. To achieve this, each workload is run in the following switch environments:

***Baseline*:** switches employ flow-level hashing

***Priority*:** in addition to *Baseline*, switches employ priority ingress and egress queues

***FC*:** in addition to *Baseline*, switches employ flow control

***Priority+PFC*:** in addition to *Priority*, switches perform priority flow control

DeTail: in addition to *Priority+PFC*, switches perform priority-aware adaptive load balancing

In the *Baseline* and *Priority* environment (i.e., environments without flow control mechanisms), a TCP timeout of 10ms is used as suggested by prior work [32, 12]. For all the other environments, the timeout is set to 50ms as discussed in Section 6.3.

These microbenchmarks are intended to highlight the advantages of DeTail across a wide range of traffic environments. In all the presented results, we focus on the 99th percentile completion time to highlight the reduction of the tail. Later, we present the results of synthetic web-facing datacenter workloads.

8.1.1 Microbenchmarks

In this section, we analyze DeTail with an all-to-all query workload - each server in the simulated datacenter generates queries to a random destination server according to a Poisson distribution. In each query, a TCP connection is used to send a request of 1460 bytes (i.e., full sized packet) and receive a response of a specific number of bytes. The size of these response (henceforth called *query size*) is randomly chosen to be 2KB, 8KB, or 32KB with equal probability. These sizes extend past the range of query traffic common in datacenter networks [12]. We select randomly from discrete sizes to enable more effective analysis of 99th percentile performance.

Bursty Workload: Network traffic is traditionally characterized as being bursty in nature. Hence we start by evaluating DeTail on a bursty workload. Every 50ms interval, each server generates a burst of queries for a duration varying from 2.5ms to 12.5ms (different burst sizes). During each of these bursts, each server generates queries to random destinations for the specified duration at an average rate of 10,000 queries / second. We consider all flows to be of the same priority level.

Figure 5 illustrates the effectiveness of DeTail in reducing the tail, by presenting the distribution of completion times of 8KB flows under a burst duration of 12.5ms. While the *Baseline* case may have a 99th percentile completion time of 85ms (more than 4.5 times the 50th percentile of 18ms), DeTails reduces the the 99th percentile

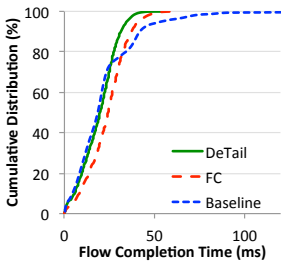
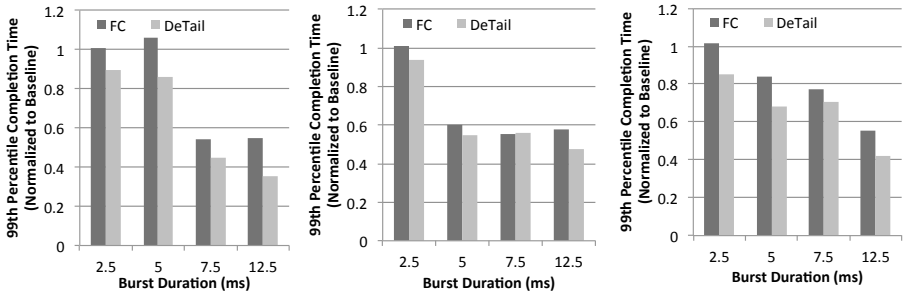


Figure 5: Distribution of completion times of the 8KB queries under a burst duration of 12.5ms



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

Figure 6: 99^{th} percentile query completion times of FC and DeTail relative to that of Baseline, for different query sizes across different burst durations.

to 40ms - a reduction of more than 50%. While flow control mechanisms is able to achieve 40% (49ms) of the reduction by completely avoid packet drops (and therefore timeouts), it does so at the cost of the median performance (compare the distribution of *FC* with *Baseline*). By also employing adaptive load balancing, DeTail ensures that the median performance remains high as well, thus illustrating the importance of the synergy between the individual mechanisms.

Figure 6 presents the 99th percentile completion time for different burst durations, normalized to that of the *Baseline* in each case. This clearly illustrates that DeTail provides between 7% and 65% reduction in the 99th percentile completion time across different burst durations and for all 3 query sizes. Reinforcing our earlier explanation, flow control mechanisms provide the most benefit in most cases (more than 40% reduction in some cases). In many cases, adaptive load balancing provides an additional benefit of up to 20% over *FC*. Note the common trend within each query size – higher the duration of the burst, greater the chance for packet drops in the *Baseline* case and hence greater the overall improvement with DeTail.

As is also visible for the steady workload that follows, there are a few cases where flow control (*FC*) increased completion times (e.g., 2KB queries under 5ms burst) over the *Baseline*. We attribute this to head-of-line blocking, which is mitigated by adaptive load balancing.

Steady Workload: We also subject DeTail to a steady workload of continuous all-to-all queries with the same query sizes as before. The average query rate is varied from 500 to 2500 queries / second, which represents load factors between 0.17 and 0.85. Figure 8.1.1 illustrates the distribution of completion times of the 8KB queries at 2000 queries / second (load factor 0.68). Unlike the bursty workload illustrated in Figure 5, there were relatively few packet drops in this steady workload. As a result, flow control mechanisms do not provide any improvement (*FC* and *Baseline* coincide with each other). Rather, adaptive load balancing provides more consistent completion times by ensuring a uniform distribution of load across all paths in the network. In Figure 8, we see that this behavior achieves about 10% to 81% improvement across the query rates for all 3 query sizes.

Note that higher rates have more improvement in the 99th percentile completion time. Specifically, at 2500 queries per second, the network is under a load high enough to cause significant packet drops, which allows *FC* to provide 20% to 25% improvement over *Baseline*. Even at such a high load and mixed query sizes, DeTail is able to sustain a 99th percentile performance of about 5.3ms for 2KB queries, a reduction of over 81% over *Baseline* of 28.7ms.

Mixed Workload: From these experiments, it is clear that both flow control and load balancing are useful mechanisms with the utility of each depending on the bursty or steady nature of the workload. We further evaluate this with a mixed workload where bursts are followed by steady periods. Similar to the bursty workload, every 50ms interval begins with a 5ms burst of queries generated at a rate of 10,000 queries/second. During the remaining 45ms, queries are generated at a lower rate to represent steady traffic. As before, each query is transmitted to a random destination and the query size is randomly chosen to be one of 2, 8, or 32KB. The steady period query rate is varied from 250 to 1000 queries/second. Figure 9 illustrates that DeTail provides between 25% to 60% reduction in 99th percentile completion times, with significant contributions from both flow control and load balancing mechanisms.

Prioritized Workload: DeTail employs multiple prioritization mechanisms to support the traffic mix common in datacenters. To evaluate the advantages of these mechanisms, we re-ran one of the mixed workloads, this time randomly assigning each flow to one of two priority levels. In Figure 10, we depict the 99th percentile completion times of *Priority* (switches employ only priority mechanisms), *Priority+PFC*, and DeTail, relative to that of *Baseline*. While the reduction in completion time of high priority flows by *Priority* is expected, DeTail

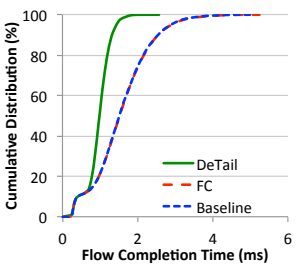
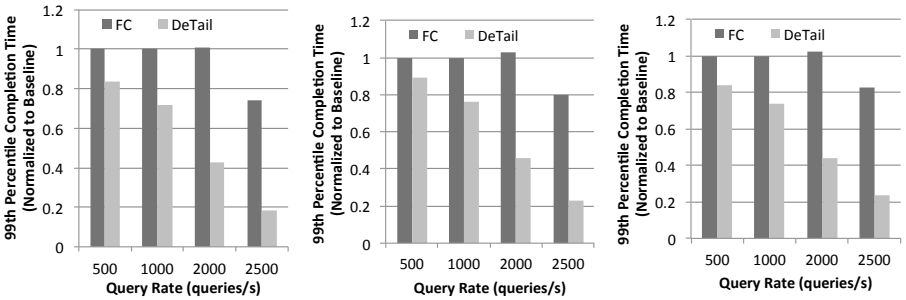
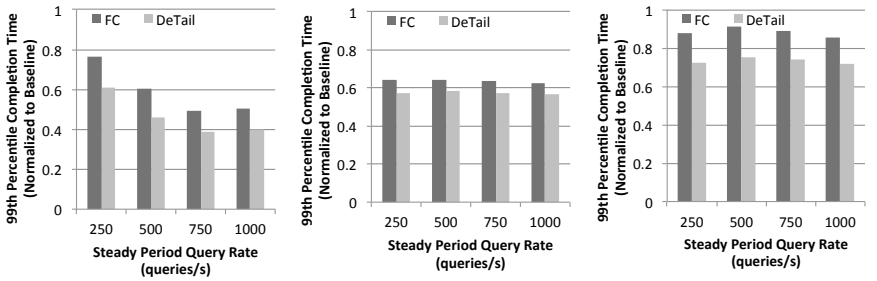


Figure 7: Distribution of completion times of 8KB queries under a steady query rate of 2000 queries/second



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

Figure 8: 99^{th} percentile query completion times of FC and DeTail relative to that of Baseline, for different query sizes across different steady query rates.



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

Figure 9: Simulation results with mixed query workload for varying data rates

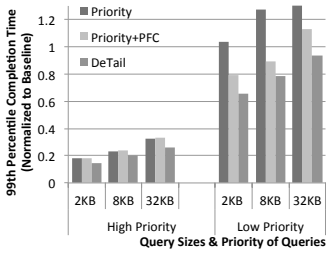


Figure 10: Simulation results with mixed query workload for varying priorities

achieves an additional 12% to 22% reduction over *Priority*. Furthermore, DeTail manages to improve the low priority flows by 7% to 35%. This further demonstrates the synergistic behavior of the mechanisms used in DeTail.

We conclude this section by summarizing the key takeaways of the microbenchmarks.

- **The benefit of the individual components of DeTail varies with workload:** Priority flow control provides the most benefit for bursty workloads while adaptive load balancing provides the most benefit for steady workloads.

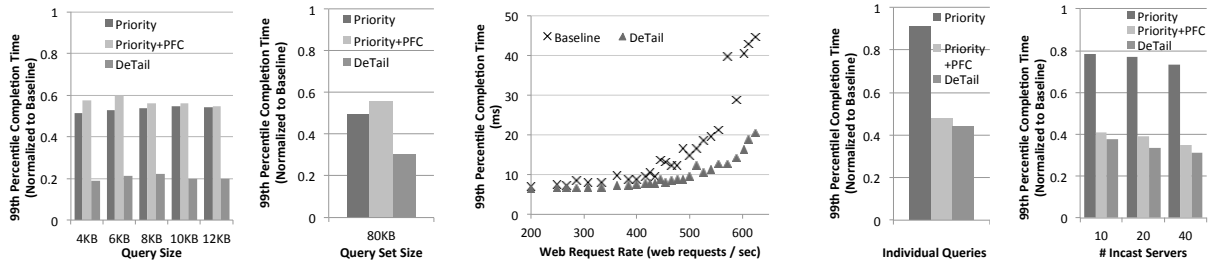
- **These mechanisms together help each other overcome the limitations of using each individually:** For example, adaptive load balancing helps overcome the head-of-line blocking that priority flow control can cause.

8.1.2 Web-facing Workload

In this section, to further illustrate the potential improvement by DeTail, we simulate and analyse two different synthetic web workloads based two different traffic patterns observed in literature [12, 34, 28] - (i) Sequential queries, and (ii) Partition/Aggregate queries. The servers in our simulated datacenter are equally divided into front-end (web-facing) servers and back-end servers. For every web request that a front-end server receives, it generates a set of data retrieval queries to randomly chosen back-end servers, either sequentially or in parallel, depending on the workload. All data-retrieval queries generated for a web-request need to complete for the web-request to be responded. The pattern of back-end queries is explained later in each of the workloads. In each case, each server also has an average of 1 long delay-insensitive low-priority flow of 1MB, which is based on the median size of long flows in datacenters [12]. As before, we focus on the 99th percentile completion time of each individual data retrieval query, as well as, the aggregate completion time of the whole set of queries (this being the minimum time that the web request will take to generate a response).

Sequential Workload: Each data retrieval query is randomly assumed to be 4, 6, 8, 10 or 12KB in size (average 8KB [1]). Each web request generates a set of 10 such queries sequentially, the total size of which is 80KB (chosen for effective analysis). The web requests are generated in the same way as the mixed workload in the microbenchmark analysis – every 50ms, each front-end server gets a burst of 800 web requests/second for 10ms, followed by a steady period of 333 web requests/second for the rest 40ms.

Figure 11(a) and Figure 11(b) shows relative performance *Priority*, *Priority+PFC*, and *DeTail*, normalized



(a) Completion time of individual data queries under mixed request workload

(b) Aggregate completion time of 10 sequential data queries

(c) Aggregate completion time of 10 sequential queries under sustained request rates

Figure 11: Simulation results of web-facing sequential query workload

(a) Completion time of individual queries

(b) Aggregate completion time of P/A queries

Figure 12: Simulation results for web-facing part.-agg. workload

to that of *Baseline*. Considering each individual data retrieval queries, it is unsurprising that prioritization is able to provide about 50% reduction for all five query sizes. But DeTail provides a further 60% reduction over Priority, which is a total of about 80% reduction. On the whole query sets, DeTail provides about 70% reduction over *Baseline*, and about 40% over *Priority*.

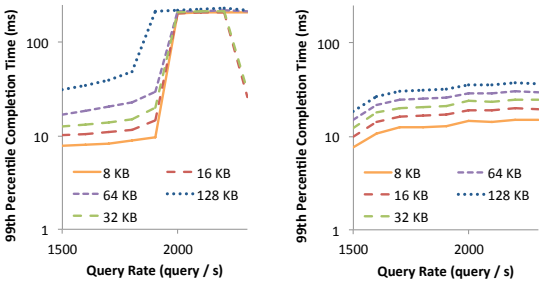
Figure 11(c) compares the aggregate completion times of 10 data retrieval queries for DeTail under sustained web request rates on each front-end server (steady load). It is evident that across a wide range of load, DeTail achieves a lower 99th percentile completion time than *Baseline*. In other words, given a deadline of 10ms, DeTail can sustain about 21% higher load than *Baseline*.

Partition/Aggregate Workload: In this workload, each web request generates a set of 2KB queries in parallel to randomly chosen 10, 20 or 40 back-end servers. Similar to the previous workload, a mixed workload of 10ms bursts at 1000 requests/second and 40ms of 333 requests / second is used.

As shown in Figure 12, DeTail achieves more than 50% reduction in the 99th percentile completion time of individual 2KB data retrival queries compared to *Baseline*, as well as, *Priority*. This translates to about 65% reduction in the aggregate completion time of the partition/aggregate queries (about 55% reduction over *Priority*).

Note the contrast between the two web workloads – while adaptive load balancing mechanism provide the maximum benefit in sequential queries, priority flow control mechanisms provide the maximum benefit in partition/aggregate workloads.

In both workloads, DeTail achieves improvement in completion times of deadline sensitive flows without harming the background flows. In fact, through effective load balancing, DeTail actually improves background flow performance by 50%.



(a) Priority

(b) DeTail

Figure 13: 99th percentile completion time in Click-based implementation

8.2 Implementation

We ran our Click-based implementation [24] on the Emulab-based Deter testbed [13]. Each of the nodes we used in the Deter cluster only has 4 experimental interfaces. So, we created a 16-server (36-node) Fat Tree topology consisting of Gigabit Ethernet links running at line rate. We designated half the servers to be back-end servers and half to be front-end servers.

To evaluate the performance of our Click-based implementation, we ran a bursty workload with different burst rates. Every second, each front-end generates a 10 ms burst of requests to randomly chosen back-end servers. The response for each of these requests is randomly chosen between 8, 16, 32, 64, and 128KB. Additionally, each front-end server is constantly engaged in a 1MB background flow to a randomly selected back-end server.

In Figure 13, we plot the 99th percentile completion times of *Priority* and DeTail. From these figures, we see that DeTail provides predictable performance, irrespective of flow size and traffic rate. In contrast, *Priority* starts to experience packet drops and timeouts at larger query rates. In these cases, DeTail provides an order of magnitude performance improvement. We recognize that a TCP implementation tuned for datacenter networks (i.e. with 10 ms min RTOs) would fare better in this environment. However, as TCP's RTOs have to be larger than

RTTs to avoid spurious retransmission, it would still perform poorly due to packet losses that occur in the absence of DeTail's mechanisms.

Before continuing, we would like to remind the reader that our Click-based implementation lacks the hardware support commonly present in hardware switches to enable mechanisms such as priority flow control. Consequently, we anticipate a DeTail-compliant switch would have performance that more closely resembles our simulations than our implementation.

9 Related Work

In this section, we discuss prior work and how it relates to our in-network, multipath-aware congestion management mechanism. To provide context for our work, we focus primarily on three areas: Internet protocols, datacenters, and HPC interconnects, discussing each in turn.

9.1 Internet Protocols

The Internet was initially designed as a series of independent layers [15] 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 [17, 14, 22] and in terms of buffer management such as RED and Fair Queuing [18, 16] were proposed. However, 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. Furthermore, all of these traditional approaches focus on single path flow traversal.

9.2 Datacenter Networks

Relevant datacenter work has focused on two areas: topologies and traffic management protocols. New topologies such as FatTrees, VL2, BCube, and DCell [10, 19, 20, 21] were proposed to increase bisection bandwidth. All of these approaches focused on 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 proposed mechanisms to improve flow completion time by reducing buffer occupancies [12]. D^3 sought to allocate flow resources based on application-specified deadlines [34]. Both of these protocols employed single path mechanisms to optimize the partition-aggregate workload. Conversely, DeTail provides multipath mechanisms designed to improve the performance of both partition-aggregate and sequential workloads, which cannot be easily parallelized and have stricter flow completion requirements.

The recent industrial effort known as Datacenter Bridging seeks to extend Ethernet to support traffic from other protocols that have different link layer assumptions [2]. DeTail leverages PFC, which was developed as a part of this effort [7]. However, as these mechanisms were designed for a different purpose, they are insufficient for meeting our performance goals. For example, they do not propose a multipath solution.

Datacenter protocols focused on spreading load across multiple paths have been proposed. Hedera performs periodic flow re-mapping of elephant flows [11]. MPTCP takes a step further, making TCP aware of multiple paths [29]. While these approaches provide multipath support, they do not perform in-network traffic management or traffic differentiation. Furthermore, these approaches traditionally focus on improving system throughput instead of reducing the flow completion time tail.

9.3 HPC Interconnects

As mentioned earlier, DeTail borrows mechanisms from HPC interconnects. Credit-based flow control has been extensively studied and deployed [9]. Adaptive load balancing algorithms such as UGAL and PAR have also been proposed [9]. 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 flow control and priority mechanisms, it does not perform adaptive load balancing. As such Infiniband suffers from the same head-of-line blocking issues that PFC mechanisms do. Host-based approaches to performing load-balancing, such as [33] have been proposed. But these host-based approaches are limited because they cannot respond in less than one RTT.

10 Conclusion

In this paper, we presented DeTail, an in-network multipath-aware congestion control mechanism designed to reduce the flow completion time tail. We demonstrated that DeTail's synthesis of existing mechanisms effectively manages congestion in the datacenter, reducing network variability. Additionally, we showed that the combination of these mechanisms is more effective than just the sum of its parts. These mechanisms assist each other to overcome the limitations of using any of them individually.

We extensively evaluated DeTail through simulation and implementation, demonstrating its ability to reduce the 99th percentile flow completion time across a wide variety of workloads. Our results verify that DeTail is an effective approach for web sites to meet their interactivity deadlines while serving richer content.

11 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. Additionally, we would like to thank Scott Shenker and Sylvia Ratnasamy for their insightful comments and suggestions.

References

[1] Average web page size septuples since 2003. http://www.websiteoptimization.com/speed/tweak/average-web-page/.

[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] Ieee standard part 3: (csma/cd) access method and physical layer specifications - section two. *IEEE Std 802.3-2008 (Revision of IEEE Std 802.3-2005)* (26 2008).

[9] ABTS, D., AND KIM, J. High performance datacenter networks: Architectures, algorithms, and opportunities. *Synthesis Lectures on Computer Architecture 6*, 1 (2011).

[10] AL-FARES, M., LOUKISSAS, A., AND VAHDAT, A. A scalable, commodity data center network architecture. In *ACM SIGCOMM Conference* (2008), SIGCOMM '08, ACM.

[11] AL-FARES, M., RADHAKRISHNAN, S., RAGHAVAN, B., HUANG, N., AND VAHDAT, A. Hedera: Dynamic flow scheduling for data center networks. In *NSDI Symposium* (2010).

[12] ALIZADEH, M., GREENBERG, A., MALTZ, D. A., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. Data center tcp (dctcp). In *ACM SIGCOMM Conference* (2010), SIGCOMM '10, ACM.

[13] 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 *In TRIDENTCOM* (2006).

[14] BRAKMO, L. S., O'MALLEY, S. W., AND PETERSON, L. L. Tcp vegas: new techniques for congestion detection and avoidance. In *ACM SIGCOMM Conference* (1994), SIGCOMM '94, ACM.

[15] CLARK, D. The design philosophy of the darpa internet protocols. In *ACM SIGCOMM Conference* (1988), SIGCOMM '88, ACM.

[16] DEMERS, A., KESHAV, S., AND SHENKER, S. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM Conference* (1989), SIGCOMM '89, ACM.

[17] FLOYD, S., AND HENDERSON, T. The newreno modification to tcp's fast recovery algorithm, 1999.

[18] FLOYD, S., AND JACOBSON, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw. 1* (August 1993).

[19] 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 *ACM SIGCOMM Conference* (2009), SIGCOMM '09, ACM.

[20] 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 *In SIGCOMM* (2009).

[21] 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 *Proceedings of the ACM SIGCOMM 2008 conference on Data communication* (2008), SIGCOMM '08, ACM.

[22] JACOBSON, V., AND BRADEN, R. T. Tcp extensions for long-delay paths, 1988.

[23] KOHAVI, R., AND LONGBOTHAM, R. Online experiments: Lessons learned, September 2007. http://exp-platform.com/Documents/IEEEComputer2007 OnlineExperiments.pdf.

[24] KOHLER, E., MORRIS, R., CHEN, B., JANNOTTI, J., AND KAASHOEK, M. F. The click modular router. *ACM Trans. Comput. Syst. 18* (August 2000).

[25] LU, G., GUO, C., LI, Y., ZHOU, Z., YUAN, T., WU, H., XIONG, Y., GAO, R., AND ZHANG, Y. Serverswitch: A programmable and high performance platform for data center networks. In *NSDI* (2011).

[26] MCKEOWN, N. White paper: A fast switched backplane for a gigabit switched router.

[27] MCKEOWN, N. The islip scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Netw. 7* (April 1999).

[28] OUSTERHOUT, J. K., AGRAWAL, P., ERICKSON, D., KOZYRAKIS, C., LEVERICH, J., MAZIÈRES, D., MITRA, S., NARAYANAN, A., ROSENBLUM, M., RUMBLE, S. M., STRATMANN, E., AND STUTSMAN, R. The case for ramclouds: Scalable high-performance storage entirely in dram. In *SIGOPS OSR* (2009), Stanford InfoLab.

[29] RAICIU, C., PLUNTKE, C., BARRE, S., GREENHALGH, A., WISCHIK, D., AND HANDLEY, M. Data center networking with multipath tcp. In *ACM HOTNETS Workshop* (2010), Hotnets '10, ACM.

[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] SHIEH, A., KANDULA, S., GREENBERG, A., KIM, C., AND SAHA, B. Sharing the data center network. In *NSDI* (2011).

[32] 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 *Proc. ACM SIGCOMM* (Aug. 2009).

[33] 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 *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid* (2007), CCGRID '07, IEEE Computer Society.

[34] WILSON, C., BALLANI, H., KARAGIANNIS, T., AND ROWTRON, A. Better never than late: meeting deadlines in datacenter networks. In *ACM SIGCOMM Conference* (2011), SIGCOMM '11, ACM.

Notes

[1]We do not consider jumbo Ethernet frames.

[2]PFC is only defined for 10 GigE. We use 1 GigE in this paper for manageable simulation times. We base our PFC response time on the time specified for Pause frames. This is appropriate because 10GigE links are given the same amount of time to respond to PFC messages are they are to Pause frames.