Deconstructing Datacenter Packet Transport

Mohammad Alizadeh†, Shuang Yang†, Sachin Katti†,
Nick McKeown†, Balaji Prabhakar†, and Scott Shenker‡

†Stanford University ‡U.C. Berkeley / ICSI
{alizade, shyang, skatti, nickm, balaji}@stanford.edu shenker@icsi.berkeley.edu

Abstract

We present, pFabric, a minimalistic datacenter fabric design that provides near-optimal performance in terms of completion time for high-priority flows and overall network utilization. pFabric's design eliminates nearly all buffering on switches (switches have only ∼20KB of buffering per port), requires almost no congestion control and uses only simple mechanisms at each switch. Specifically, switches are only required to locally and greedily decide what packets to schedule and drop according to priorities in the packet header and do not maintain any flow state or rate estimates. Rate-control is almost unnecessary, all flows start at line-rate and only slow down in the extreme case of congestion collapse. We show via simulations using realistic workloads and topologies that this simple design achieves near optimal flow completion times and network utilization.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Performance

Keywords

Datacenter fabric, Flow scheduling, Quality of service

1. INTRODUCTION

Datacenter network fabrics are required to provide high fabric utilization and effective flow prioritization. High fabric utilization is of course a natural goal, however flow prioritization in datacenters has unique and stringent requirements. Specifically, datacenters are used to support large-scale web applications that are generally architected to generate a large number of short flows to exploit parallelism, and the overall response time is dependent on the latency of each of the short flows. Hence providing near-fabric latency to the short latency-sensitive flows while maintaining high fabric utilization is an important requirement in datacenters.

Recent research [2, 3, 11, 7, 12, 9] has begun to address this problem. One line of work [2, 3] tries to adapt congestion control algorithms to provide flow prioritization. The basic idea is to strive to keep queues empty through a variety of mechanisms (pacing, ECN based feedback, throttling the elephant flows etc) so that latency sensitive flows see small buffers and consequently small latencies. While they improve latency, such implicit techniques are fundamentally constrained because they can never precisely estimate the right flow rates to use so as to schedule flows to minimize latency. Furthermore, due to the bursty nature of traffic, keeping network queues empty is challenging and requires carefully designed rate-control and hardware packet pacing at the end-hosts, and trading off network utilization [3].

Having recognized the above limitation, subsequent work [11, 7] *explicitly* assigns a sending rate to each flow in order to schedule the flows according to some notion of urgency. The assigned rates are typically computed in the network based on flow information such as the deadline or estimated completion time. While this approach can potentially provide very good performance, it is rather complex and challenging to implement in practice. The difficulty is in requiring the network to explicitly assign a rate to each flow to perform scheduling. Doing this efficiently requires maintaining fairly accurate flow information (size, deadline, desired rate, round-trip time, etc) at switches, and also coordination among switches to identify the bottleneck for each flow and avoid under-utilization [7]. This is a major burden, both in terms of communication overhead and requisite state at switches, particularly in the highly dynamic datacenter environment where flows arrive and depart

Hotnets '12, October 29–30, 2012, Seattle, WA, USA.

at high rates and the majority of flows last only a few RTTs [6, 4].

Despite these various problems, we should be clear that these proposed mechanisms provide vastly improved performance for datacenter networks. Our goal here is not to try to beat these proposals in a quantitative race, but instead to deconstruct datacenter transport and examine which fundamental pieces of functionality are necessary for good performance. In other words, we are asking: *What is the simplest fabric design that provides high fabric utilization and effective flow prioritization?*

We present pFabric, a minimalistic datacenter fabric design that meets these requirements. pFabric's entire design consists of the following:

- End-hosts put a single number in the header of every packet that encodes their priority (e.g. the deadline, remaining flow size). The priority is set independently by each flow, and no coordination is required across flows or hosts.
- Switches are simple, they have very small buffers (max 22.5KB per port) and decide which packets to accept into the buffer and which ones to schedule strictly according to the packet's priority number. When a new packet comes in, if the buffer is full and the incoming packet has lower priority than all buffered packets, it is dropped. Else, the lowest priority packet in the buffer is dropped and replaced with the incoming packet. When transmitting, the switch sends the packet with the highest priority. Thus each switch operates independently in a greedy and local fashion.
- Rate control is minimal, all flows start at line-rate and throttle their sending rate according to a simple control law only if they see high and persistent loss rates approaching congestion collapse. Hence rate control need not be accurate and is almost trivial to implement.

pFabric thus requires no flow state at the switches, no feedback from the network, no rate calculations and assignments at the switches, no high-speed large switch buffers, nor complex congestion control mechanisms at the end-host.

The key insight behind our design is to decouple the mechanisms for flow prioritization from rate control. Specifically, the priority based dropping and scheduling mechanisms at each switch ensure that each switch schedules flows in order of their priorities at any time. Further, as we will discuss in §2, the local and greedy decisions made by each switch lead to an approximately optimal flow scheduling decision across the entire fabric. Once flow prioritization is handled, rate control's goal is much simpler: to avoid persistently high packet drop rates. Hence, the rate control design gets correspondingly simpler, flows start at line rate and throttle only if bandwidth is being wasted due to excessive drops. The above design coupled with the fact that datacenter networks have small bandwidth delay products (BDP) of around 10–15 packets allows us to keep the buffers on each switch very small. Switches only need to keep the highest priority packets that will fill one BDP, since with the aggressive sending rates a lesser priority packet that is dropped will get retransmitted by the time the switch buffer drains.

We evaluate our design with detailed packet-level simulations in ns2 [8] using two widely used datacenter workloads: one that mimics a typical data mining workload [6] and one that mimics a web application workload [2]. We show that pFabric achieves near-optimal fabric latency for the short flows. Further, the network utilization is kept close to the optimal depending on the load on the network. pFabric reduces the latency for short flows compared to traditional TCP and DCTCP by more than 7–11× and 2.5–4× respectively at the mean, and more than 33–69× and 7–17× respectively at the 99th percentile.

2. DESIGN

pFabric's key design insight is a principled decoupling of flow prioritization from rate control. Specifically, if we take a broader view of recent work in this space, in effect these proposals are attempting to do flow prioritization by precisely controlling rates of flows. For example, DCTCP [2] is using feedback from the network and adjusting the rate control loop to ensure that high-priority small flows do not see large queues of packets from long flows. Similarly D3 [11] and PDQ [7] are explicitly controlling rates for individual flows at the switches to perform flow prioritization (thus ensuring that small flows have a reserved link capacity and long flows are appropriately throttled). A similar line of thinking can be traced across other recent proposals.

We argue that rate control is a complex and ineffective technique to perform precise flow prioritization. Specifically, precisely computing rates to prioritize flows requires accurate information on the number of flows, their relative sizes, the paths they traverse and so on. Since such information cannot be scalably collected, prior approaches have to resort to making informed guesses at the end-hosts on what rates to use based on network congestion feedback. Or they have to keep dynamic flow state at the switches, and compute and signal per-flow rates continuously as flows arrive and leave. The first approach leads to sub-optimal performance, while the second approach increases network complexity and signaling overhead. Further, it requires the support of complex pacing mechanisms at the end-hosts to actually be able to send at the rates specified.

pFabric decouples these goals. We design a simple

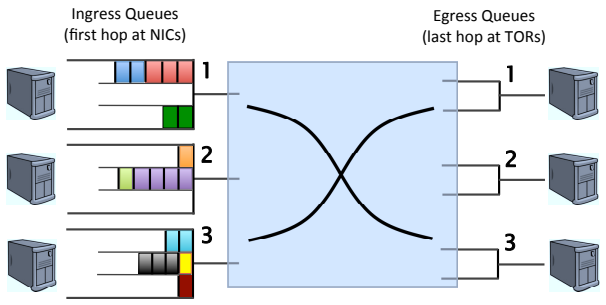


Figure 1: Conceptual view of flow scheduling over a datacenter fabric.

switch based technique that takes care of flow prioritization and this ends up significantly simplifying rate control. Throughout this section, similar to prior work [11, 7] we will assume that packet headers carry one number that indicates the priority of that flow (e.g. the deadline for the flow, remaining flow size etc).

2.1 Flow prioritization

Our conceptual contribution in designing the flow prioritization technique is a viewpoint that abstracts out the entire fabric as one giant switch. Specifically, the datacenter fabric typically consists of two or three layers of switches in a Fat-tree or Clos topology. Instead of focusing on the individual switches, the whole fabric can be abstracted as one giant switch that interconnects the servers as shown in Fig. 1. The ingress queues into the fabric switch are at the NICs and the egress queues out of the fabric switch are at the last-hop TOR switch.[1] Each ingress port (source NIC) has some flows destined to various egress ports. It is convenient to view these as organized in 'virtual output queues' at the ingress as shown in Fig. 1. For example, the red and blue flows at ingress 1 are destined to egress 1, while the green flow is destined to egress 3.

In this context, transport over the datacenter fabric can essentially be thought of as scheduling flows over the back-plane of a giant switch. The problem is to find the best schedule for sending packets over the fabric in accordance with some objective. Many objectives are possible (e.g. throughput maximization, max-min fairness), but for the datacenter context a natural goal is minimizing average flow completion time (FCT) [7]. Since datacenter workloads are dominated by large numbers of short flows, minimizing average FCT will ensure that the short, high-priority flows see very low latency. Further, scheduling disciplines for minimizing average FCT are theoretically tractable and well understood (see below). However, we stress that our conceptual model can be used to design transport for other objectives (such as minimizing missed deadlines) as well.

The optimal algorithm for minimizing average FCT when scheduling over a single link is the *Shortest Remaining Processing Time (SRPT)* policy, which always schedules the flow that has the least work remaining. However, we are not scheduling over a single link, but rather over an entire fabric with a set of links connecting the ingress and egress queues. Unfortunately, a simple universally optimal policy does not exist for simultaneously scheduling multiple links. In fact, even under the simplifying assumption that the fabric core can sustain 100% throughput and only the ingress and egress access links are potential bottlenecks, the scheduling problem for minimizing the average completion time is equivalent to the NP-hard *sum-multicoloring problem* [5].

Our key observation is that a greedy algorithm which schedules flows across the fabric switch based on size — in non-decreasing order of size in a maximal manner such that at any time a flow is blocked if and only if either its ingress port or its egress port is busy serving a different (smaller) flow — is theoretically proven to provide at least a 2-approximation to the optimal scheme (see [5] for more details). In fact, in practice, the performance is even closer to optimal (§3). Thus the takeaway is that a greedy scheduler that prioritizes small flows over large flows end-to-end across the fabric can provide near-ideal average flow completion time.

2.2 pFabric switches

To realize the above greedy scheduling discipline, our switch design consists of two simple mechanisms:

- **Priority scheduling:** Whenever a port is idle, the packet with the highest priority buffered at the port is dequeued and sent out.
- **Priority dropping:** Whenever a packet arrives to a port with a full buffer, if it has priority less than or equal to the lowest priority packet in the buffer, it is dropped. Otherwise, one or more of the lowest priority packets in the buffer are dropped to make room for the new packet.

In both mechanisms, ties are broken at random.

The packet priority number can be chosen in various ways at the end-hosts to approximate different scheduling disciplines. The above discussion has been focused on minimizing the average FCT, and to achieve that that objective, end-hosts can set the priority for each packet to be the *remaining flow size*. However if the network designer wished to minimize the number of missed flow deadlines, she can emulate the *Earliest Deadline First (EDF)* policy with pFabric by setting the packet priority to be the *remaining time until the deadline*, quantized in some unit (e.g. microseconds).

[1]We assume the receiver NICs run at line rate and are not bottlenecks. Therefore, we do not consider the receiver NICs as part of the fabric.

Implementation complexity. Priority scheduling and dropping are easy to implement since pFabric switches only buffer about a bandwidth-delay product worth of packets at each port (typically 15–22.5KB in a 10Gbps fabric). Traditionally, datacenter switches use nearly 10–30× more buffering per port. Thus, with a 22.5KB buffer, in the worst-case of minimum size 64B packets, we have 51.2ns to find the highest/lowest of at most ∼350 numbers. A straight-forward implementation of this as a binary tree requires just 9 ($\log_2(350)$) clock cycles. Today, this is trivial in 40nm or 28nm ASICs.

Note that our switches do not keep any other state, nor are they expected to provide feedback, nor do they perform rate computations. Further, the significantly smaller buffering requirement lowers the overall switch design complexity and die area [3].

2.3 Rate control

What about rate control? Because pFabric switches send and drop the 'right' packets, the need for rate control is minimal. In particular, we do not need rate control to prevent spurious packet drops due to bursts, as can occur, for example, in Incast [10] scenarios. Such events only impact the *lowest* priority packets at the time, which can quickly be retransmitted before the bottleneck switch runs out of higher priority packets, essentially without impacting performance (see §2.4 for details). Further, we need not worry about keeping queue occupancy small in switches to control latency. Since packets are dequeued based on priority, even if large queues could form in the fabric (which they can't, because buffers are small by design), there would be no impact on the latency for high-priority traffic.

However, there is one corner case where a limited form of rate control is necessary. Specifically, whenever a packet traverses multiple hops only to be dropped at a downstream link, some bandwidth is wasted on the upstream links that could have been used to transmit other packets. The bandwidth inefficiency is most severe when the load is high and multiple elephant flows collide at a downstream link. For example, if two elephant flows sending at line rate collide at the last-hop access link, half the bandwidth they consume is wasted. If such high loss rates persist, it would eventually lead to congestion collapse in the fabric. Note that packet drops at the ingress are not an issue, since they do not waste any bandwidth in the fabric.

Fortunately, designing a rate control scheme that only has to prevent congestion collapse (maintain an acceptable loss rate) is simple; after all, that was the initial goal of the simplest TCP designs. In fact, we disable all of the newer mechanisms in TCP such as fast retransmit, dup-ACKs, and timeout estimation and return TCP to its roots. Flows start with a large window size (at least a BDP) and the window undergoes normal additive increase each round-trip time. Decreases only happens upon a timeout. No other congestion signal is used. This simple scheme is sufficient to avoid congestion collapse, and can in fact be deployed right now after disabling the appropriate "features" from TCP stacks.

2.4 Why this works

Ideal flow scheduling is achieved so long as at each switch port and at any time, one of the highest priority packets that needs to traverse the port is available to be scheduled. Now, when a packet is dropped, by design it has the lowest priority among all buffered packets. Hence, even if it were not dropped, its 'turn' to be scheduled would not be until at least *all* the other buffered packets have left the switch.[2] Therefore, the packet can safely be dropped as long as it is retransmitted before all the existing packets depart the switch. This can easily be achieved if the buffer size is equal to (or larger than) one bandwidth-delay product and hence takes at least an end-to-end RTT to drain, and the rate control allows the flow to transmit another packet within a RTT. The rate control outlined above which keeps flows at line-rate most of the time easily meets this requirement.

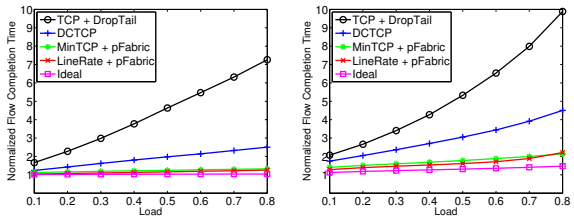
2.5 Extension to multiple traffic classes

For simplicity, our description of pFabric assumed that the same priority assignment scheme is used for all traffic. However, in practice, datacenter fabrics are typically shared by a variety of applications or traffic classes with different requirements and a single priority structure may not always be appropriate. This can easily be handled by operating the pFabric priority scheduling and dropping mechanisms *within individual traffic classes* in an hierarchical fashion. Essentially, conventional high-level quality of service is provided for traffic classes based on user-defined policy (e.g., a soft-real time application is given a higher weight than a data-mining application), while pFabric provides near-optimal scheduling of individual flows in each class to minimize flow completion times.

3. EVALUATION

In this section we conduct a preliminary evaluation of pFabric using detailed packet-level simulations in ns2 [8] of a realistic datacenter topology with empirical models of workloads derived from recent studies [6, 2]. We show that pFabric achieves near-ideal flow completion time metrics, both at the mean and at the 99th percentile for the small flows. Further, this performance is achieved in almost all the cases with no congestion con-

[2]Note that the packet's turn may end up being even further in the future if higher priority priority packets arrive while it is waiting in the queue.



(a) Data mining workload (b) Web search workload

Figure 2: Overall average flow completion time. The results are normalized with respect to the best possible completion time for each flow size.

trol algorithm, flows can simply start at line rate and stay there until their last packet is acknowledged.

Topology. We simulate a 54 port three-layered fat-tree which is a very common datacenter topology [1]. The fabric consists of 45 6-port switches organized in 6 pods and has full-bisection bandwidth. All links are 10Gbps and the end-to-end round-trip latency (across 6 hops) is $\sim 12\mu$s.

Workloads. We simulate dynamic workloads where flows are initiated between randomly chosen source and destination servers. We use two flow size distributions that are based on measurements from production datacenters reported in the literature. The first distribution is from a cluster running large data mining jobs [6]. The second distribution is from a datacenter supporting web search [2]. Both of these workloads have a diverse mix of small and large flows with heavy-tailed characteristics. In the data mining workload, more than 80% of the flows are less than 10KB and more than 80% of all bytes are in flows larger than 100MB. The web search workload is less skewed: Over 95% of all bytes come from the 30% of the flows that are 1–20MB. Flow arrivals are according to a Poisson process. We vary the flow arrival rate to simulate traffic with overall load levels between 10%–80%.

Schemes. We compare the following schemes:

(i) TCP+DropTail: TCP with standard 225KB (150 packets) DropTail queues.

(ii) DCTCP: DCTCP with 225KB queues and 22.5KB ECN marking threshold.

(iii) MinTCP+pFabric: MinTCP and pFabric with 22.5KB queues. MinTCP is the following stripped down variant of TCP: 12 packet initial window size; no fast-retransmission; no retransmission timeout estimation, RTO is fixed at 40μs.

(iv) LineRate+pFabric: No rate control and pFabric with 22.5KB queues. Flows constantly send at the NIC line rate.

(v) Ideal: Ideal end-to-end SRPT scheduling as described in §2.1. A central scheduler with a full view of all flows preemptively schedules existing flows in nondecreasing order of size, and in a maximal manner. Only the access link capacity constraints are considered and the internal links are ignored (see Fig. 1). This is a flow-level simulation (not packet-level) conducted in Matlab.

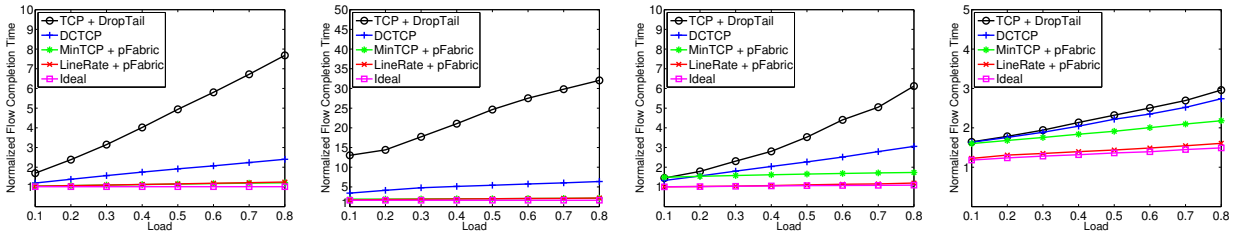
For all pFabric simulations, we assume that packets contain the remaining flow size in the priority field approximate SRPT scheduling.

Note: We have simulated a variety of load-balancing schemes, including *Equal-Cost-Multipathing (ECMP)*, and *packet spraying*, where each switch blindly sprays packets among all shortest-path next hops in round-robin fashion. We found that both TCP and DCTCP perform better with packet spraying (after fast retransmissions are disabled to cope with re-ordering), as compared to standard TCP/DCTCP with ECMP. Hence to make a fair comparison, we compare pFabric with the above modified versions of TCP and DCTCP. We have also tuned the $minRTO$ to find the best settings for the retransmission timer for all schemes and found that $minRTO = 200\mu s$ for 225KB queues with TCP and DCTCP, and $minRTO = 40\mu s$ for 22.5KB queues with LineRate+pFabric and MinTCP+pFabric achieve the best performance.

Analysis: Overall Average FCT. The overall average flow completion times (FCT) for the two workloads are shown in Fig. 2 as we vary the load. We normalize each flow's completion time to the *best possible value* — given the 10Gbps link speed and the round-trip time — for a flow of that size. We observe that for both workloads, MinTCP+pFabric and LineRate+pFabric achieve near ideal average FCT and significantly outperform TCP and DCTCP. The average FCT is reduced by a factor of $2.5 - 11\times$ especially at high load.

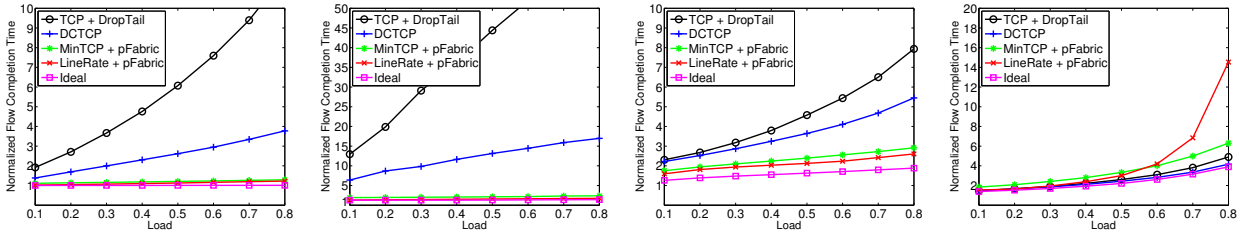
Analysis: FCT breakdown. To understand the benefits further, we show the FCT statistics across three flow size bins (0, 100KB], (100KB, 10MB], and (10MB, ∞) in Figures 3 and 4 for both workloads. We plot the average FCT for each bin and also the 99th percentile FCT for the smallest flow size bin for which the tail latency is important.

Data mining workload. MinTCP+pFabric and LineRate+pFabric both have near-optimal FCT for the smallest flows (0, 100KB] even under heavy loads as well as at the high percentiles. For example, the average (normalized) FCT with LineRate+pFabric is 1.04–1.25 and the 99th percentile is 1.68–2.18. MinTCP+pFabric is similar: 1.04–1.21 at the mean, and 1.90–2.12 at the 99th percentile. Both are nearly an order of magnitude better than TCP and DCTCP. LineRate+pFabric is also very good for the medium and large flows and is always within $\sim$10% of the Ideal scheme. MinTCP+pFabric is slightly worse for the large flows, but still performs better than TCP and DCTCP. This is because MinTCP conservatively reduces its window size and goes through slowstart after each packet drop.



(a) (0, 100KB]: Avg (b) (0, 100KB]: 99th prctile (c) (100KB, 10MB]: Avg (d) (10MB, ∞): Avg

Figure 3: Data mining workload: Normalized FCT statistics across different flow sizes.



(a) (0, 100KB]: Avg (b) (0, 100KB]: 99th prctile (c) (100KB, 10MB]: Avg (d) (10MB, ∞): Avg

Figure 4: Web search workload: Normalized FCT statistics across different flow sizes.

Web search workload. The improvement with pFabric for the small flows is even larger in the Web search workload. For instance, at 80% load, the 99th percentile FCT for the (0,100KB] flows is only 1.75 with LineRate+pFabric, while it is 16.97 and 66.03 respectively for DCTCP and TCP. For the large flows (> 10MB), however, we do observe a slowdown with pFabric at high loads particularly with LineRate+pFabric. This is because of persistent high loss rates for the elephant flows that leads to wasted bandwidth. It demonstrates that while in majority of scenarios, rate-control is unnecessary, some form of it is required in extreme cases. In fact, MinTCP+pFabric is much better than LineRate+pFabric for the large flows at high load. It still exhibits a (∼20–28%) slowdown compared to TCP and DCTCP though. This is because the large flows are de-prioritized with pFabric. Since in the Web search workload, more than 65% of the total load is from flows smaller than 10MB, the de-prioritization impacts the large flows.[3] The takeaway is that a minimal rate control scheme that essentially operates flows at line rate most of the time, but uses a stripped down version of TCP such as MinTCP at very high load and persistently high packet loss would provide the best performance across all scenarios.

4. CONCLUSION

This paper deconstructs the different aspects of datacenter packet transport (flow prioritization and network utilization), and shows that by designing very simple mechanisms for these two goals separately, we can realize a minimalistic datacenter fabric design that achieves near-ideal performance. Further, it shows how surprisingly, buffers or even congestion control are largely unnecessary in datacenters. Our next step is to prototype and evaluate the performance of pFabric in a deployed datacenter testbed with real traffic, as well as develop a general theoretical model that provides a principled analysis of how close to ideal pFabric is.

[3]Note that this effect is not noticeable in the Data mining workload, since less then 5% of the traffic is from flows smaller than 10MB.

5. REFERENCES

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

[2] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proc. of SIGCOMM*, 2010.

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

[4] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload analysis of a large-scale key-value store. In *Proc. of SIGMETRICS*, pages 53–64, New York, NY, USA, 2012. ACM.

[5] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, R. Salman, and H. Shachnai. Sum multicoloring of graphs. *J. Algorithms*, 37(2):422–450, Nov. 2000.

[6] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *Proc. of SIGCOMM*, 2009.

[7] C.-Y. Hong, M. Caesar, and P. B. Godfrey. Finishing Flows Quickly with Preemptive Scheduling. In *Proc. of SIGCOMM*, 2012.

[8] The Network Simulator NS-2. `http://www.isi.edu/nsnam/ns/`.

[9] B. Vamanan, J. Hasan, and T. N. Vijaykumar. Deadline-Aware Datacenter TCP (D2TCP). In *Proc. of SIGCOMM*, 2012.

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

[11] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: meeting deadlines in datacenter networks. In *Proc. of SIGCOMM*, pages 50–61. ACM, 2011.

[12] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. H. Katz. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In *Proc. of SIGCOMM*, 2012.