RepFlow: Minimizing Flow Completion Times with Replicated Flows in Data Centers

Hong Xu*, Baochun Li†
henry.xu@cityu.edu.hk, bli@eecg.toronto.edu
* Department of Computer Science, City University of Hong Kong
† Department of Electrical and Computer Engineering, University of Toronto

***Abstract*—Short TCP flows that are critical for many interactive applications in data centers are plagued by long flows and head-of-line blocking in switches. Hash-based load balancing schemes such as ECMP aggravate the matter and result in long-tailed flow completion times (FCT). Previous work on reducing FCT usually requires custom switch hardware and/or protocol changes. We propose RepFlow, a simple yet practically effective approach that replicates each short flow to reduce the completion times, without any change to switches or host kernels. With ECMP the original and replicated flows traverse distinct paths with different congestion levels, thereby reducing the probability of having long queueing delay. We develop a simple analytical model to demonstrate the potential improvement. Further, we conduct NS-3 simulations and Mininet implementation and show that RepFlow provides 50%–70% speedup in both mean and 99-th percentile FCT for all loads, and offers near-optimal FCT when used with DCTCP.**

I. Introduction

Data centers run many interactive services and applications that impose stringent requirements on the transport fabrics. They often partition computation into many small tasks, distribute them to thousands of machines, and stitch the responses together to return the final result [4], [39]. Such partition-aggregation workflows generate a large number of short query and response flows across many machines, and demand that short flows have low latency in order to provide soft real-time performance to users. More importantly, the tail latency also needs to be low since the request completion time depends on the slowest flow.

TCP is the dominant transport protocol in data centers [4]. Flow completion times (FCT) for short TCP flows are poor: FCT can be as high as tens of milliseconds while in theory they could finish in 10–20 microseconds with 1G or 10G interconnects. The reason is that these flows are often queued behind bursts of packets from long flows of other workloads (e.g. backup and data mining), a phoenomenon known as head-of-line blocking. The situation is even worse with hash-based flow-level load balancing schemes such as ECMP [3]. ECMP is agnostic to congestion, does not differentiate between short and long flows, and may assignment many long flows to the same path causing flash congestions and long-tailed FCT even when the network is lightly loaded [3], [39]. We measure the round-trip times (RTT) between two small instances in Amazon EC2's us-west-2c zone every 1 second for 100K samples as a rough estimation of FCT. The newer EC2 data centers are known to have many equal-cost paths between given pairs of instances [30]. Fig. 1 and Fig. 2 confirm the long-tailed distribution: While mean RTT is only 0.5ms, the 99-th percentile RTT is 17ms.

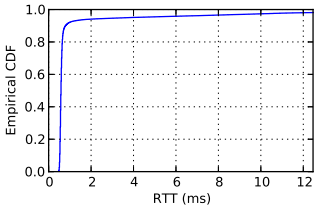


Fig. 1: CDF of RTT between two small instances in EC2 us-west-2c.

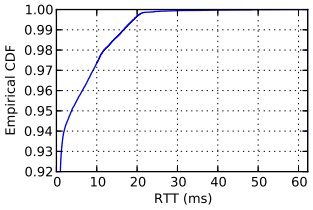


Fig. 2: Long tail of the RTT distribution.

Recent research has proposed many new transport designs to reduce FCT of short flows. Broadly speaking, the central idea is to reduce the short flows' queueing delay via adaptive ECN-based congestion control [4], explicit flow rate assignment [19], deadline-aware scheduling [26], [34], [37], priority queueing [6], and cross-layer redesign [6], [39]. While effective, they do require modifications to switches and operating systems, making it difficult to deploy them in a large-scale data center with a large number of servers and switches.

Our goal is to design a practical and effective data center transport scheme that provides low latency for short flows both on average and in the 99-th percentile, and can be readily deployed in current infrastructures. To this end, we present RepFlow, a simple data center transport design. RepFlow directly uses existing TCP protocols deployed in the network. The only difference with RepFlow is that it replicates each short TCP flow by creating another TCP connection to the receiver, and sending identical packets for both flows. The application uses the first flow that finishes the transfer. Flow replication can be easily implemented as libraries or middleware at the application layer. Thus RepFlow requires no change to switches, the network stack, or operating systems, and maintains TCP's robustness for throughput intensive traffic. It can also be used with other data center transport protocols such as DCTCP [4] to further improve performance as we will show later.

The key insight behind RepFlow is the observation that

978-1-4799-3360-0/14/$31.00 ©2014 IEEE

multi-path diversity, which is readily available with high bisection bandwidth topologies such as Fat-tree [2], is an effective means to combat performance degradation that happens in a random fashion. Flash congestion due to bursty traffic and imperfect load balancing happen randomly in any part of the network at any time. As a result, congestion levels on different paths are statistically independent. In RepFlow, the replicated and original flow are highly likely to traverse different paths, and the probability that both experience long queueing delay is much smaller. RepFlow targets general clusters running mixed workloads, where short flows typically represent a very small fraction ($< 5\%$) of overall traffic according to measurements [4], [15]. Thus the replication overhead and impact on throughput for long flows are rather mild.

It is important to note that RepFlow works with ECMP per-flow load balancing, and differs from multipathing schemes such as MPTCP [30] and packet spraying [12] that split a flow across multiple paths. Traffic is asymmetric and dynamic, especially considering link failures and external traffic that originates or terminates outside of the data center. When the paths used by a flow have different loads, out-of-order packets interact negatively with TCP. Splitting a flow hardly reduces latency for short flows, though it improves throughput for long flows. ECMP is also widely used in current data centers, reducing the implementation overhead of RepFlow.

We evaluate RepFlow with queueing analysis, packet-level simulations in NS-3, and Linux kernel-based implementation using Mininet [17]. We develop a simple M/G/1 queueing model to model mean and tail FCT. Our model shows that the diversity gain of replication can be understood as a reduction in the effective traffic load seen by short flows, which leads to significantly improved queueing delay and FCT. Our evaluation uses two data center traffic traces: one that mimics a web search workload [4] and one that mimics a typical data mining workload [15]. NS-3 simulations with a 16-pod 1,024-host Fat-tree, and experiments with a 4-pod Fat-tree on Mininet show that RepFlow achieves 50%–70% speedup in both mean and 99-th percentile FCT even for loads as high as 0.8 compared to TCP. When it is feasible to use advanced transport protocols such as DCTCP [4], RepFlow offers competitive performance compared to state-of-the-art clean slate approaches such as pFabric [6]. The overhead to the network is negligible, and long flows are virtually not affected. Thus we believe it is a lightweight and effective approach that requires minimal implementation efforts with salient FCT reductions.

II. Related Work

Motivated by the drawbacks of TCP, many new data center transport designs have been proposed. We briefly review the most relevant prior work here. We also introduce some additional work that uses replication in wide-area Internet, MapReduce, and distributed storage systems for latency gains.

Data center transport. DCTCP [4] and HULL [5] use ECN-based adaptive congestion control and appropriate throttling of long flows to keep the switch queue occupancy low in order to reduce short flows' FCT. D^3 [37], D2TCP [34], and PDQ [19] use explicit deadline information to drive the rate allocation, congestion control, and preemptive scheduling decisions. DeTail [39] and pFabric [6] present clean-slate designs of the entire network fabric that prioritize latency sensitive short flows to reduce the tail FCT. All of these proposals require modifications to switches and operating systems. Our design objective is different: we strive for a simple way to reduce FCT without any change to TCP and switches, and can be readily implemented at layers above the transport layer. RepFlow presents such a design with simple flow replication that works with any existing transport protocol.

Replication for latency. Though seemingly naive, the general idea of using replication to improve latency has gained increasing attention in both academia and industry for its simplicity and effectiveness. Google reportedly uses request replications to rein in the tail response times in their distributed systems [11]. Vulimiri et al. [36] argue for the use of redundant operations as a general method to improve latency in various systems, such as DNS, databases, and networks. In the context of wide-area Internet, [38] argues for the latency benefit of having multiple wide-area transit links in a multi-cloud CDN deployment scenario. Replication has also been recently used in MapReduce [7] and storage systems [33] to mitigate straggling jobs. As a related technique, Mitzenmacher's "power of two choices" work [25] proposes for a request to randomly choose two servers, and queue at the one with less requests to achieve good load balancing without a global view of the system.

III. RepFlow: Motivation and Design

A. Motivation

In today's data center networks based on a Fat-tree or Clos topology [4], [15], many paths of equal distance exist between a given pair of end-hosts. Equal-cost multi-path routing, or ECMP, is used to perform flow-level load balancing. When a packet arrives at a switch, ECMP picks an egress port uniformly at random among equal-cost paths based on the hash value of the five-tuple in the packet header. All packets of the same flow then follow a consistent path.

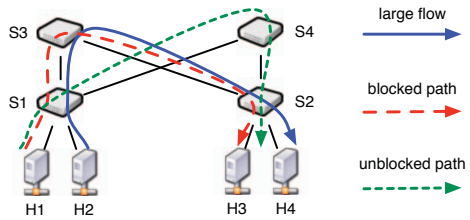


Fig. 3: A large flow transmits from H2 to H4 following the path shown in the solid line. If the short flow from H1 to H3 takes the path S1–S3–S2, it will queue behind packets of the large flow in the links S1–S3 and S3–S2. If it takes the path S1–S4–S2, there is no head-of-line blocking since all ports of this path are empty. ECMP will randomly hash a short flow to one of the two paths.

Due to hash collisions in ECMP, flows are often routed on the same path. Short flows then have to wait until packets of long flows are processed and suffer from head-of-line

blocking. Consider a toy example shown in Fig. 3. The topology resembles one pod of a 4-pod Fat-tree network. A host under switch S1(S2) has two paths to each host under switch S2(S1). There is a persistent long flow from H2 to H4, taking the path S1–S3–S2. Now H1 starts to send short flows of 10 packets continuously to H3. ECMP randomly hashes short flows with 0.5 probability to the path S1–S3–S2, creating head-of-line blocking and long FCT. We conduct an experiment in Mininet [17] using exactly the same setup, and observe the mean FCT is 10x worse with the long flow as shown in the following table (more on Mininet in Sec. VI).

Scenario	Mean FCT	99-th percentile FCT
without a long flow	0.0135s	0.0145s
with a long flow	0.175s	0.490s
with a long flow and replication	0.105s	0.212s

TABLE I: Mininet experiment results of the toy example shown in Fig. 3. Each link is 50Mb with 1ms delay. Short flows of 10 packets are sent continuously from H1 to H3 with ECMP. With the large flow, short flows suffer from 10x worse FCT. Replication dramatically improves mean and tail FCT by over 50% in this case.

ECMP creates serious problems. Yet it also provides a promising solution—multi-path diversity, which motivates RepFlow. In the toy example it is obvious that the path S1–S4–S2 has much shorter queueing delay. By replicating a short flow and making sure the two flows have distinct five-tuples, one copy of it will traverse this path and improve FCT significantly. From the same Mininet experiment we observe over 50% improvement with simple replication in this case, as shown in Table I. It is in general difficult to choose the right path for short flows beforehand in a data center network with a large number of flows, not to mention the latency overhead. Replication removes this need by opportunistically utilizing the less congested paths.

B. Design

RepFlow uses flow replication to exploit multi-path diversity. It does not modify the transport protocol, and thus works on top of TCP as well as any other TCP variants, such as DCTCP [4] and D2TCP [34]. On the high level, there are several design decisions we need to make. First, which short flow should we replicat? We mandate that flows less than or equal to 100KB are considered short flows, and are replicated to achieve better latency. This threshold value is chosen in accordance with many existing papers [6], [19], [26], [39]. Second, we need to decide when to replicate the flows. One might argue that we should only replicate when flows are experiencing long queueing delays to reduce the replication overhead. However the extremely short duration of these flows makes such a reactive approach too slow to remedy the situation. In its current design, RepFlow proactively replicates each and every short flow from the very beginning to achieve the best latency. As we will show in Sec. V-E, the overhead of doing so is negligible, thanks to the well-known fact that short flows only account for a tiny fraction of total bytes in production networks [6], [20]. Finally, we replicate exactly once for simplicity, though more replication is possible.

RepFlow can be implemented in many ways. The simplest is to create two TCP sockets when a short flow arrives, and send the same packets through two sockets. This is also our current implementation. Since data centers run a large number of applications, it is preferable to provide RepFlow as a general library or middleware for any application to invoke [1]. For example one may implement RepFlow as a new transport abstraction in Thrift, a popular RPC framework used by companies like Facebook [32]. We are currently investigating this option. Another possibility is to implement RepFlow at the transport layer, by modifying TCP so that short flows are marked and automatically replicated with two independent subflows. This approach provides transparency to applications, at the cost of requiring kernel upgrades. In this space, RepFlow can be incorporated into MPTCP [31] with its multi-path support.

RepFlow lends itself to many implementation choices. Regardless of the detail, it is crucial to ensure path diversity is utilized, i.e. the five-tuples of the original and replicated flow have to be different (assuming ECMP is used). In our implementation we use different destination port numbers for this purpose.

IV. Analysis

Before we evaluate RepFlow at work using simulations and experiments, in this section we present a queueing analysis of flow completion times in data centers to theoretically understand the benefits and overhead of replication.

A. Queueing Model

A rich literature exsits on TCP steady-state throughput models for both long-lived flows [24], [27] and short flows [18]. There are also efforts in characterizing the completion times of TCP flows [10], [23]. See [10] and references therein for a more complete literature review. These models are developed for wide-area TCP flows, where RTTs and loss probabilities are assumed to be constants. Essentially, these are open-loop models. The data center environment, with extremely low fabric latency, is distinct from the wide-area Internet. RTTs are largely due to switch queueing delay caused by TCP packets, the sending rate of which in turn are controlled by TCP congestion control reacting to RTTs and packet losses. This closed-loop nature makes the analysis more intriguing [29].

Our objective is to develop a simple FCT model for TCP flows that accounts for the impact of queueing delay due to long flows, and demonstrates the potential of RepFlow in data center networks. We do not attempt to build a fine-grained model that accurately predicts the mean and tail FCT, which is left as future work. Such a task is potentially challenging because of not only the reasons above, but also the complications of timeouts and retransmissions [28], [35], switch buffer sizes [8], [23], etc. in data centers.

We construct our model based on some simplifying assumptions. We abstract one path of a data center network as a M/G/1 first-come-first-serve (FCFS) queue with infinite buffer. Thus we do not consider timeouts and retransmissions. Flows arrive following a Poisson process and have size $X \sim F(\cdot)$. Since

TCP uses various window sizes to control the number of in-flight packets, we can think of a flow as a stream of bursts arriving to the network. We assume the arrival process of the bursts is also Poisson. One might argue that the arrivals are not Poisson as a burst is followed by another burst one RTT later (implying that interarrival times are not even i.i.d). However queueing models with general interarrival time distributions are difficult to analyze and fewer results are available [14]. For tractability, we rely on the commonly accepted M/G/1-FCFS model [6], [8]. We summarize some key notations in the table below. Throughout this paper we consider (normalized) FCT defined as the flow's completion time normalized by its best possible completion time without contention.

TABLE II: Key notations.

M	maximum window size (64KB, 44 packets)
S_L	threshold for long flows (100KB, 68 packets)
$F(\cdot), f(\cdot)$	flow size CDF and PDF
$\rho \in [0,1)$	overall traffic load
W	queueing delay of the M/G/1-FCFS queue
k	initial window size in slow-start

For short flows, they mostly stay in the slow-start phase for their life time [8], [10], [13], [23]. Their burst sizes depend on the initial window size k. In slow-start, each flow first sends out k packets, then $2k$, $4k$, $8k$, etc. Thus, a short flow with X packets will be completed in $\log_2(X/k+1)$ RTTs, and its normalized completion time can be expressed as

$$FCT_X = \sum_{i=1}^{\log_2(X/k+1)} W_i/X + 1, \quad (1)$$

assuming link capacity is 1 packet per second.

For long flows that are larger than S_L, we assume that they enter the congestion avoidance phase immediately after it arrives [24], [27]. They continuously send bursts of a fixed size equal to the maximum window size M (64KB by default in Linux). A large flow's FCT is then

$$FCT_X^L = \sum_{i=1}^{X/M} W_i/X + 1, X \geq S_L. \quad (2)$$

B. Mean FCT Analysis

We now analyze the mean FCT for short flows in TCP. On average, each burst sees the expected queueing delay $E(W)$. Thus according to (1), the mean FCT of a flow with X packets is

$$E[FCT_X] = \log_2(X/k+1)\frac{E[W]}{X} + 1.$$

The mean FCT for short flows less than S_L is

$$E[FCT] = E[W]\int_0^{S_L} \frac{\log_2(x/k+1)}{x}\frac{f(x)}{F(S_L)}\mathrm{d}x + 1. \quad (3)$$

The mean queueing delay of a M/G/1-FCFS queue with load ρ is obtained with the famous Pollaczek-Khintchine formula [16]:

$$E[W] = \frac{\rho}{2(1-\rho)}\frac{E[B^2]}{E[B]} = \frac{\rho M}{2(1-\rho)}, \quad (4)$$

where B denotes the burst size as opposed to the flow size. Since most of the bytes are from long flows, almost all bursts arrive to the queue are of a fixed size M, and $E[B^2]/E[B] = M$. Therefore we have

$$E[FCT] = \frac{\rho M}{2(1-\rho)}\int_0^{S_L} \frac{\log_2(x/k+1)}{x}\frac{f(x)}{F(S_L)}\mathrm{d}x + 1. \quad (5)$$

The mean FCT for short flows depends on the load of the network and the flow size distribution. Many data center operators opt for an increased initial window size to reduce latency in the slow-start phase [13]. So we use $k = 12$ packets [6], [13] throughout the paper. Using a flow size distribution from a production data center running web search workloads [4], Fig. 4 plots the FCT with varying load.

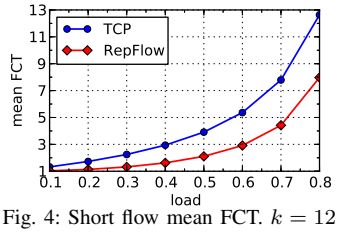


Fig. 4: Short flow mean FCT. $k = 12$ packets, flow size distribution from the web search workload [4].

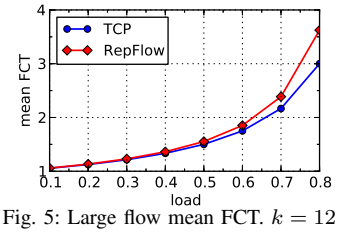


Fig. 5: Large flow mean FCT. $k = 12$ packets, flow size distribution from the web search workload [4].

We now turn our attention to RepFlow, and obtain its mean FCT expression. For each short flow, RepFlow sends two identical copies by initiating two TCP connections between the same end-points. With ECMP, each flow is transmitted along different paths and experiences different congestion levels. We model this as having two independent queues with independent arrival processes and the same load ρ. When a short flow arrives, it enters both queues and get serviced, and its completion time is based on the faster queue.

Without replication, each short flow sees a queue of load ρ, i.e. the network is busy with probablility ρ when the flow enters, and idle with probablility $1-\rho$. Now with replication, each queue's load is slightly increased from ρ to $(1+\epsilon)\rho$, where

$$\epsilon = \frac{\int_0^{S_L} xf(x)\mathrm{d}x}{E[X]}.$$

ϵ is the fraction of total bytes from short flows, and is usually very small (less than 0.1 [4], [15], [20]). Since the two queues are independent, a short flow will find the network busy only when both queues are busy with probability $(1+\epsilon)^2\rho^2$, and idle with probability $1-(1+\epsilon)^2\rho^2$. In other words, each flow is effectively serviced by a virtual queue of load $(1+\epsilon)^2\rho^2$. Thus, the mean FCT for RepFlow is simply

$$E[FCT_{rep}] = \frac{(1+\epsilon)^2\rho^2 M}{2(1-(1+\epsilon)^2\rho^2)}\int_0^{S_L} \frac{\log_2(x/k+1)}{x}\frac{f(x)}{F(S_L)}\mathrm{d}x + 1. \quad (6)$$

For small $\epsilon \leq 0.1$, $(1+\epsilon)^2\rho^2$ is much smaller than ρ. As ρ increases the difference is smaller. However the factor $\rho/(1-\rho)$ that largely determines the queueing delay $E[W]$ and FCT is very sensitive to ρ in high loads, and a small decrease of load

leads to significant decrease in FCT. In the same Fig. 4, we plot FCT for RepFlow with the same web search workload [4], where 95% of bytes are from long flows, i.e. $\epsilon = 0.05$. Observe that RepFlow is able to reduce mean FCT by a substantial margin compared to TCP in all loads.

Our analysis reveals that intuitively, the benefit of RepFlow is due to a significant decrease of effective load experienced by the short flows. Such a load reduction can be understood as a form of multi-path diversity discussed earlier as a result of multi-path network topologies and randomized load balacing.

At this point one may be interested in understanding the drawback of RepFlow, especially the effect of increased load on long flows. We now perform a similar FCT analysis for long flows. For a large flow with $X > S_L$ packets, substitute (4) to (2) yields

$$E[FCT^L] = \frac{\rho M}{2(1-\rho)}\frac{X}{M \cdot X} + 1 = \frac{\rho}{2(1-\rho)} + 1. \quad (7)$$

The mean FCT for long flows only depends on the traffic load. With RepFlow, load increases to $(1+\epsilon)\rho$, and FCT becomes

$$E[FCT^L_{rep}] = \frac{(1+\epsilon)\rho}{2(1-(1+\epsilon)\rho)} + 1, \quad (8)$$

For long flows, load only increases by ϵ, whereas small flows see a load decrease of $1-(1+\epsilon)^2\rho$. long flows are only mildly affected by the overhead of replication. Fig. 5 plots the mean FCT comparison for long flows. As we shall see from simulations and implementations results in Sec. V-E and Sec. VI the performance degradation is almost negligible even in high loads.

C. 99-th Percentile FCT Analysis

To determine the latency performance at the extreme cases, such as the 99-th percentile FCT [4], [5], [19], [39], we need the probability distribution of the queueing delay, not just its average. This is more difficult as no closed form result exists for a general M/G/1 queueing delay distribution. Instead, we approximate its tail using the effective bandwidth model [21], which gives us the following

$$P(W > w) \approx e^{-w\frac{2(1-\rho)}{\rho}\cdot\frac{E[X]}{E[X^2]}} = e^{-w\frac{2(1-\rho)}{\rho M}}. \quad (9)$$

This equation is derived in the extended version of [8]. Setting (9) equal to 0.01, we obtain the 99-th percentile queueing delay $\tilde{W}$:

$$\tilde{W} = \ln 10 \cdot \frac{\rho M}{1-\rho} = 2\ln 10 \cdot E[W]. \quad (10)$$

Recall short flows finish in $\log_2(X/k+1)$ rounds. If a flow experiences a queueing delay of $\tilde{W}$ in one round, the total FCT will be guaranteed to hit the 99-th percentile tail[1]. Thus, we can

[1]We cannot say that the delay is $\tilde{W}$ for all rounds, which happens with probability $0.01^{\log_2(X/k+1)}$, i.e. much smaller than 0.01 even for small number of rounds.

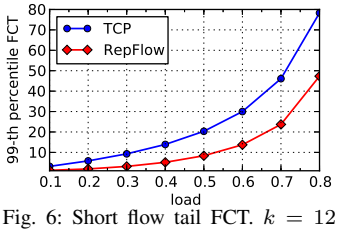


Fig. 6: Short flow tail FCT. $k = 12$ packets, flow size distribution from the web search workload [4].

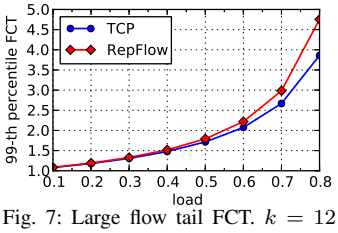


Fig. 7: Large flow tail FCT. $k = 12$ packets, flow size distribution from the web search workload [4].

approximate the 99-th percentile FCT for short flows using $\tilde{W}$ as:

$$\tilde{FCT} = E[W]\int_0^{S_L} \frac{\log_2(x/k+1) - 1 + 2\ln 10}{x}\frac{f(x)}{F(S_L)}\mathrm{d}x + 1$$
$$= E[W]\cdot N + 1. \quad (11)$$

By the same token, we can calculate the 99-th percentile FCT for short flows under RepFlow.

$$\tilde{FCT}_{rep} = E[W_{rep}]\cdot N + 1, \quad (12)$$
$$\text{where } E[W_{rep}] = \frac{(1+\epsilon)^2\rho^2 M}{2(1-(1+\epsilon)^2\rho^2)}.$$

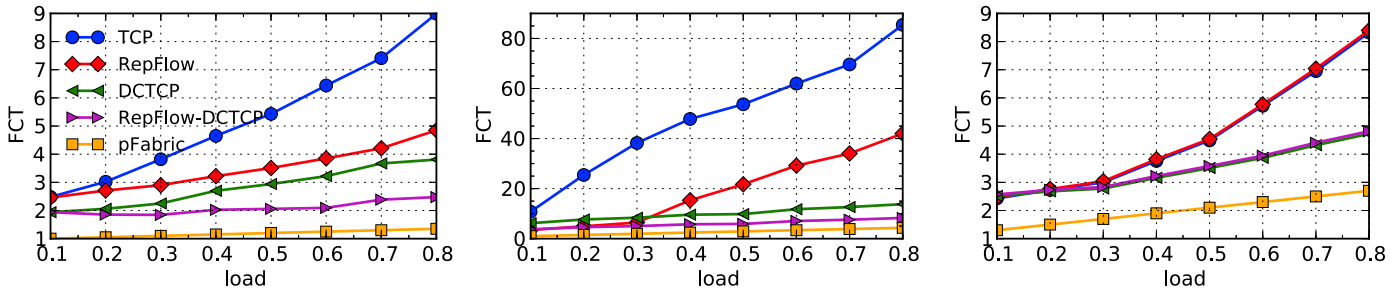
From (11) and (12) we can see that the tail FCT depends critically on the queueing delay, which is determined by the traffic load ρ. Therefore RepFlow provides better tail latency in addition to better average latency, since it reduces the effective load seen by the short flows. Fig. 6 shows the numerical results using the web search workload where we observe ∼40%–70% tail FCT improvement.

According to queueing theory, the most likely reason for the extreme events such as 99-th percentile FCT to happen is that for some time all inter-arrival times are statistically smaller than usual [9]. This has an intuitive interpretation in our problem. Recall that our queue resembles a path of the data center network connecting many pairs of end-hosts. The arrival process to our queue is in fact a composition of many Poisson arrival processes generated by the hosts, with ECMP controlling the arrival rates. While the aggregate arrival rate on average is λ, at times the queue would see the instantaneous arrival rates from individual hosts much higher than usual due to hash collisions in ECMP, resulting in the tail FCT.

The tail FCT analysis for long flows can be similarly derived as follows.

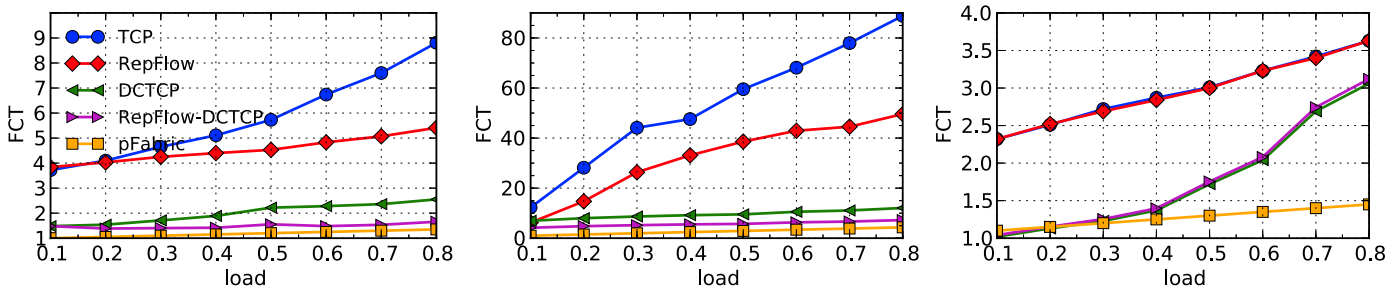
$$\tilde{FCT}^L = E[FCT^L] + (2\ln 10 - 1)E[W]\cdot P, \quad (13)$$
$$\tilde{FCT}^L_{rep} = E[FCT^L_{rep}] + (2\ln 10 - 1)E[W^L_{rep}]\cdot P, \quad (14)$$
$$\text{where } P = \int_{S_L}^{\infty}\frac{1}{x}\frac{f(x)}{1-F(S_L)}\mathrm{d}x,$$
$$E[W^L_{rep}] = \frac{(1+\epsilon)\rho M}{2(1-(1+\epsilon)\rho)}.$$

Fig. 7 shows the numerical results. long flows enjoy better tail FCT performance compared to short flows, since their transmission lasts for a long time and is not sensitive to long-tailed queueing delay. Again observe that RepFlow does not penalize long flows.



(a) (0,100KB]: Avg (b) (0,100KB]: 99-th percentile (c) (100KB, ∞): Avg

Fig. 8: FCT breakdown for different flows with a 16-pod Fat-tree and the web search workload [4] in NS-3.



(a) (0,100KB]: Avg (b) (0,100KB]: 99-th percentile (c) (100KB, ∞): Avg

Fig. 9: FCT breakdown for different flows with a 16-pod Fat-tree and the data mining workload [15] in NS-3.

D. Summary

We summarize our analytical findings. Short flows' mean and tail FCT depend critically on queueing delay, and the factor $\frac{\rho}{1-\rho}$ assuming a M/G/1-FCFS queue. Using replication, they have much less probability of entering a busy queue, and the effective load they experience is greatly reduced. This confirms the intuition that RepFlow provides path diversity gains in data center networks. RepFlow is expected to have speedup around 40%–70% as numerical results show. The negative impact on large flow is very mild, because from long flows' perspectives, load only increases slightly.

V. Experimental Evaluation

We now evaluate RepFlow using packet-level simulations in the NS-3 simulator. Building on this, we show how RepFlow performs in a realistic small-scale implementation running Linux kernel code based on Mininet [17] in the next section.

A. Methodology

Topology: We use a 16-pod Fat-tree as the network topology [2], which is commonly used in data centers. The fabric consists of 16 pods, each containing an edge layer and an aggregation layer with 8 switches each. Each edge switch connects to 8 hosts. The network has 1,024 hosts and 64 core switches. There are 64 equal-cost paths between any pair of hosts at different pods. Each switch is a 16-port 1Gbps switch, and the network has full bisection bandwidth. The end-to-end round-trip time is $\sim 32\mu s$. ECMP is used as the load balancing scheme.

Benchmark workloads: We use empirical workloads to reflect traffic patterns that have been observed in production data centers. We consider two flow size distributions. The first is from a cluster running web search [4], and the second is from a data center mostly running data mining jobs [15]. Both workloads exhibit heavy-tailed characteristics with a mix of small and long flows. In the web search workload, over 95% of the bytes are from 30% of flows larger than 1MB. In the data mining workload, 95% of all bytes are from ∼3.6% flows that are larger than 35MB, while more than 80% of flows are less than 10KB. Flows are generated between random pairs of hosts following a Poisson process with load varying from 0.1 to 0.8 to thoroughly evaluate RepFlow's performance in different traffic conditions. We simulate 0.5s worth of traffic at each run, and ten runs for each load. The entire simulation takes around 900+ machine-hours.

B. Schemes Compared

TCP: Standard TCP-New Reno is used as the baseline of our evaluation. The initial window is set to 12KB, and switches use DropTail queues with a buffer size of 100 packets. These are standard settings used in many studies [6], [34].

RepFlow: Our design as described in Sec. III. All flows less than 100KB are replicated. Other parameters are the same as TCP.

DCTCP: The DCTCP protocol with ECN marking at DropTail queues [4]. Our implementation is based on a copy of the source code we obtained from the authors of D2TCP [34]. The ECN marking threshold is set to 5%. Other parameters are set following [4].

RepFlow-DCTCP: This is RepFlow on top of DCTCP. As discussed in Sec. III RepFlow can work with any TCP variant. We use this scheme to demonstrate RepFlow's ability to further reduce FCT for networks that adopt specialized data center transport such as DCTCP.

pFabric: This is the state-of-the-art approach that offers near-optimal performance in terms of minimizing flow completion times [6]. pFabric assigns higher priority to flows with less remaining bytes to transfer, encodes the flow priority in packet headers, and modifies switches so they schedule packets based on flow priority. Thus short flows are prioritized with near-optimal FCT. Our implementation is based on a copy of the source code we obtained from the authors of the paper [6]. We follow [6] and set the DropTail queue size to be 36KB at each switch port for best performance.

C. RepFlow on TCP

We first evaluate RepFlow on TCP. RepFlow significantly reduces the mean and 99-th percentile FCT for short flows in both workloads compared to TCP. Fig. 8 and Fig. 9 show the FCT for different flows in different workloads as we vary the load. With the web search workload, RepFlow reduces mean FCT by ∼40%–45% for short flows for loads from 0.4 to 0.8. The improvement in tail FCT is more salient. RepFlow can be ∼10x faster than TCP when the load is low (<0.4), and over 60% in all other loads. The data mining workload yields qualitatively similar observations.

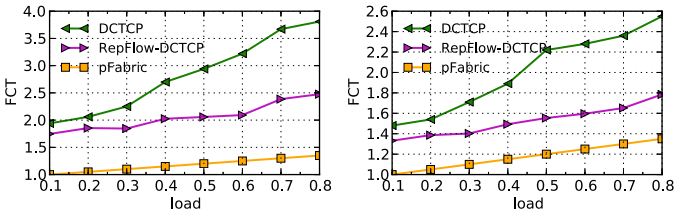
The results demonstrate the advantage of replication in harvesting the multi-path diversity, especially when the load is relatively low which is usually the case in production networks [20]. The tail FCT reduction is more substantial in low loads, while the mean FCT reduction is more significant in high loads. The reason is that when the load is low, most short flows finish quickly, with a few exceptions that hit the long tail. Since path diversity is abundant with low loads, RepFlow can easily reduce the tail latency by a large margin. When the load is higher, almost every short flow experiences some queueing delay resulting in longer average FCT. RepFlow is thus able to provide diversity gains even in the average case for most short flows. The results also corroborate our analysis and numerical results in Sec. IV.

We also look at the impact of replication on long flows. From Fig. 8c and 9c we can see that long flows suffer negligible FCT increase. Thus in a realistic network with production workloads, RepFlow causes little performance degradation and perform better than our analysis predicts. Note that long flows in the data mining workload has better FCT than those in the web search workload. This is because the data mining workload is more skewed with elephant flows larger than 1MB, while in the web search workload there are many "medium" flows of size between 100KB and 1MB. These medium flows also suffer from queueing delay especially in the slow-start phase, resulting in the larger FCT in Fig. 8c.

D. RepFlow on DCTCP

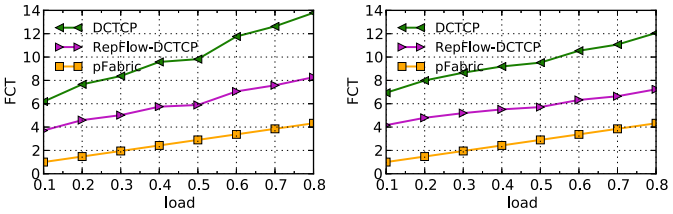
RepFlow's full potential is realized with specialized data center transport such as DCTCP [4]. For data centers that have already adopted DCTCP, RepFlow can be readily utilized just as with regular TCP. We observe in Fig. 8 and Fig. 9 that DCTCP improves FCT significantly compared to TCP, especially the 99-th percentile FCT for short flows. We show the mean and tail FCT for DCTCP, RepFlow-DCTCP and pFabric only in Fig. 10 and 11 for better contrasts. Observe that RepFlow-DCTCP reduces mean FCT by another 40% in both workloads, and is only ∼30% slower than pFabric for the data mining workload. In terms of 99-th percentile FCT, DCTCP, RepFlow-DCTCP and pFabric all provide almost an order of magnitude reduction compared to TCP. RepFlow cuts down another ∼35% tail FCT on top of DCTCP, and performs close to the near-optimal pFabric [6]. In general short flows perform better in the data mining workload than in the web search workload. This is because in the data mining workload it is less likely that some long flows are transmitting concurrently on the same port, implying less contention with short flows.

The improvements of RepFlow with DCTCP are less significant than with TCP. The reason is that by using ECN marking, DCTCP keeps the queue length very small most of the time, yielding less path diversity for RepFlow to exploit.



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

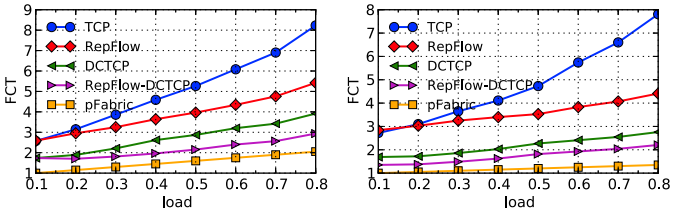
Fig. 10: Mean FCT for short flows with a 16-pod Fat-tree in NS-3. Note the different ranges of the y-axis in the plots.



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

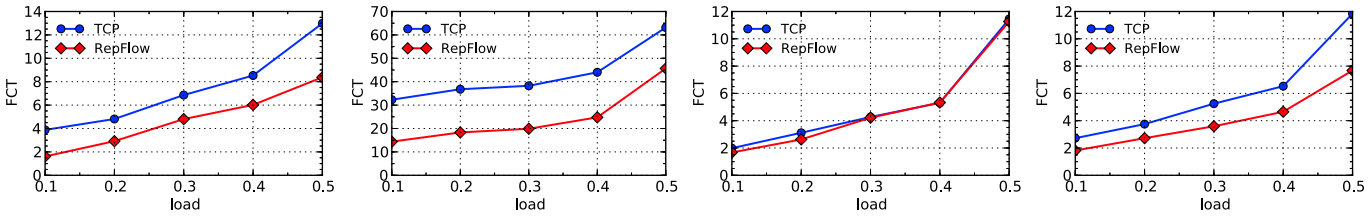
Fig. 11: 99-th percentile FCT for short flows with a 16-pod Fat-tree.

Overall, Fig. 12 shows the average FCT across all flows for all schemes. RepFlow improves TCP by ∼30%–50% in most cases. RepFlow-DCTCP improves DCTCP further by ∼30%, providing very close-to-optimal FCT compared to state-of-the-art pFabric.



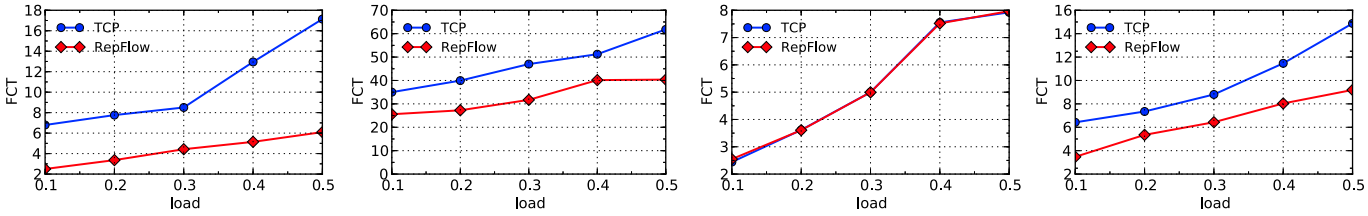
(a) Web search workload (b) Data mining workload

Fig. 12: Mean FCT for all flows.



(a) (0, 100KB]: Avg (b) (0, 100KB]: 99-th percentile (c) (100KB, ∞): Avg (d) Overall: Avg

Fig. 13: Implementation on Mininet with a 4-pod Fat-tree and the web search workload [4].



(a) (0, 100KB]: Avg (b) (0, 100KB]: 99-th percentile (c) (100KB, ∞): Avg (d) Overall: Avg

Fig. 14: Implementation on Mininet with a 4-pod Fat-tree and the data mining workload [15].

E. Replication Overhead

Replication clearly adds more traffic to the network. In this section we investigate the overhead issue of RepFlow. We calculate the percentages of extra bytes caused by replicated flows in both workloads for all loads as shown in Table III. Since short flows only generate a tiny fraction of all bytes in both workloads, not surprisingly replication does not incur excessive overhead to the network. The overhead for the DCTCP implementation is essentially the same and we omit the results.

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
3.45%	2.78%	3.13%	3.38%	3.29%	3.47%	3.22%	3.27%

(a) Web search workload

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
1.41%	1.18%	2.13%	1.38%	1.33%	1.07%	1.12%	1.09%

(b) Data mining workload

TABLE III: Overhead of RepFlow in NS-3 simulation.

To summarize, RepFlow achieves much better FCT for short flows compared to TCP with minimal impact on long flows. The improvements are not as significant as pFabric [6]. This is expected since RepFlow only opportunistically utilizes the less congested path without being able to reduce queueing delay. However, since RepFlow does not require switch hardware or kernel changes, it is an effective and practical approach to the imminent problem of reducing FCT. On the other hand when it is feasible to implement RepFlow on top of advanced transports such as DCTCP, RepFlow performs competitively against pFabric.

VI. Implementation on Mininet

We implement RepFlow on Mininet, a high-fidelity network emulation framework built on Linux container based virtualization [17]. Mininet creates a virtual network, running real Linux kernel, switch and application code on a single machine. Its scale is smaller than production data center networks due to the single-machine CPU limitation (tens of Mbps link bandwidth compared to 1Gbps). Mininet has been shown to faithfully reproduce implementation results from [3], [4] with high fidelity [17], and has been used as a flexible testbed for networking experiments [22].

Our implementation follows the design in Sec. III. We run socket-based sender and receiver programs as applications on virtual hosts in Mininet. Each virtual host runs two receivers, one for receiving regular flows and the other for replicated flows, in separate threads and listening on different ports. A flow is created by spawning a sender thread that sends to the regular port of the receiving host, and if it is a short flow another sender thread sending to the other port. The replicated flow shares the same source port as the original flow for idenfitification purpose. We implement RepFlow on top of TCP-New Reno in Mininet 2.0.0 on a Ubuntu 12.10 LTS box.

We use a 4-pod Fat-tree with 16 hosts connected by 20 switches, each with 4 ports. Each port has a 50-packet buffer. We set link bandwidth to 20Mb and delay to 1ms, which is the minimum delay Mininet supports without high-precision timers. RiplPOX is installed as the controller on switches to support ECMP. The entire experiment takes ∼6 hours to run on an EC2 c1.xlarge instance with 8 cores. We observe that Mininet becomes unstable when the load exceeds 0.5, possibly due to its scalability limitation. Thus we only show results for loads from 0.1 to 0.5.

Fig. 13 and 14 show the results. Fig. 13a and 14a show RepFlow has ∼25%–50% and 50%–70% mean FCT improvements in the web search and data mining workload, respectively. The improvement in tail FCT is around 30% in most cases for both workloads and is smaller than the NS-3 simulation results. The reason is two-fold. First there are fewer equal-cost paths in the 4-pod Fat-tree than the 16-pod Fat-tree in the simulation, implying less path diversity for RepFlow. Second, in the Mininet implementation, each sender thread of

the replicated flow is forked after the original flow. The time difference due to virtualization overhead is non-negligible for short flows (∼1ms). Thus it less likely for the replicated flows to finish faster, reducing the potential of RepFlow. We believe in a real implementation without virtualization this is unlikely to be an issue. Fig. 13c and 14c confirm that long flows are not affected by replication. Overall the implementation results are in line with simulation results.

VII. Concluding Remarks

We presented the design and evaluation of RepFlow, a simple approach that replicates short TCP flows to reap path diversity in data centers to minimize flow completion times. Analytical and experimental results demonstrate that it reduces mean and tail FCT significantly with no changes to existing infrastructures. We believe flow replication is an intuitive approach to combat unpredictable performance degradations, including but not limited to slow and long-tailed FCT. Our work is a first step in this direction. Our next step is to prototype RepFlow as a general application library running on TCP and DCTCP, and evaluate its benefits for real applications in a deployed data center. Ultimately, this may pave the path for practical use of RepFlow at scale.

Acknowledgments

We thank Balajee Vamanan and Shuang Yang for providing their simulation code in the papers [34] and [6] for DCTCP and pFabric, respectively. We also thank David Maltz, Matei Zaharia, Ganesh Ananthanaray, Kai Chen, and Christopher Stewart for providing helpful feedback and suggestions.

References

[1] Personal communication with David Maltz.
[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. ACM SIGCOMM*, 2008.
[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proc. USENIX NSDI*, 2010.
[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proc. ACM SIGCOMM*, 2010.
[5] 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. USENIX NSDI*, 2012.
[6] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. M. B. Prabhakar, and S. Shenker. pFabric: Minimal near-optimal datacenter transport. In *Proc. ACM SIGCOMM*, 2013.
[7] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Effective straggler mitigation: Attack of the clones. In *Proc. USENIX NSDI*, 2013.
[8] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proc. ACM SIGCOMM*, 2004.
[9] O. Boxma and B. Zwart. Tails in scheduling. *SIGMETRICS Perform. Eval. Rev.*, 34(4):13–20, March 2007.
[10] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP latency. In *Proc. IEEE INFOCOM*, 2000.
[11] J. Dean. Achieving rapid response times in large online services. Berkeley AMPLab Cloud Seminar, http://research.google.com/people/jeff/latency.html, March 2012.
[12] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the impact of packet spraying in data center networks. In *Proc. IEEE INFOCOM*, 2013.
[13] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An argument for increasing TCP's initial congestion window. *ACM SIGCOMM Comput. Commun. Rev.*, 40(3):26–33, June 2010.
[14] S. Foss. *Wiley Encyclopedia of Operations Research and Management Science*, chapter The G/G/1 Queue. 2011.
[15] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *Proc. ACM SIGCOMM*, 2009.
[16] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris. *Fundamentals of Queueing Theory*. Wiley-Interscience, 2008.
[17] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. In *Proc. ACM CoNEXT*, 2012.
[18] J. Heidemann, K. Obraczka, and J. Touch. Modeling the performance of HTTP over several transport protocols. *IEEE/ACM Trans. Netw.*, 5(5):616–630, October 1997.
[19] C.-Y. Hong, M. Caesar, and P. B. Godfrey. Finishing flows quickly with preemptive scheduling. In *Proc. ACM SIGCOMM*, 2012.
[20] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of datacenter traffic: Measurements & analysis. In *Proc. IMC*, 2009.
[21] F. P. Kelly. Notes on effective bandwidths. In *Stochastic networks: Theory and applications*, pages 141–168. Oxford University Press, 1996.
[22] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Proc. USENIX NSDI*, 2013.
[23] A. Lakshmikantha, C. Beck, and R. Srikant. Impact of file arrivals and departures on buffer sizing in core routers. *IEEE/ACM Trans. Netw.*, 19(2):347–358, April 2011.
[24] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, July 1997.
[25] M. D. Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, UC Berkeley, 1996.
[26] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan. Minimizing flow completion times in data centers. In *Proc. IEEE INFOCOM*, 2013.
[27] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proc. ACM SIGCOMM*, 1998.
[28] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan. Measurement and analysis of TCP throughput collapse in cluster-based storage systems. In *Proc. USENIX FAST*, 2008.
[29] R. S. Prasad and C. Dovrolis. Beyond the model of persistent TCP flows: Open-loop vs closed-loop arrivals of non-persistent flows. In *Proc. IEEE ANSS*, 2008.
[30] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath tcp. In *Proc. ACM SIGCOMM*, 2011.
[31] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How hard can it be? designing and implementing a deployable multipath TCP. In *Proc. USENIX NSDI*, 2012.
[32] M. Slee, A. Agarwal, and M. Kwiatkowski. Thrift: Scalable cross-language services implementation. Technical report, Facebook, 2007.
[33] C. Stewart, A. Chakrabarti, and R. Griffith. Zoolander: Efficiently meeting very strict, low-latency SLOs. In *Proc. USENIX ICAC*, 2013.
[34] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter TCP (D2TCP). In *Proc. ACM SIGCOMM*, 2012.
[35] 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. ACM SIGCOMM*, 2009.
[36] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Low latency via redundancy. In *Proc. ACM CoNEXT*, 2013.
[37] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron. Better never than late: Meeting deadlines in datacenter networks. In *Proc. ACM SIGCOMM*, 2011.
[38] Z. Wu and H. V. Madhyastha. Understanding the latency benefits of multi-cloud webservice deployments. *ACM SIGCOMM Computer Communication Review*, 43(1):13–20, April 2013.
[39] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz. DeTail: Reducing the flow completion time tail in datacenter networks. In *Proc. ACM SIGCOMM*, 2012.