2013 IEEE 21st Annual Symposium on High-Performance Interconnects

TCP Pacing in Data Center Networks

Monia Ghobadi
Department of Computer Science
University of Toronto
Email: monia@cs.toronto.edu

Yashar Ganjali
Department of Computer Science
University of Toronto
Email: yganjali@cs.toronto.edu

***Abstract*—This paper studies the effectiveness of TCP pacing in a data center setting. TCP senders inject bursts of packets into the network at the beginning of each round-trip time. These bursts stress the network queues which may cause loss, reduction in throughput and increased latency. Such undesirable effects become more pronounced in data center environments where traffic is bursty in nature and buffer sizes are small. TCP pacing is believed to reduce the burstiness of TCP traffic and to mitigate the impact of small buffering in routers. Unfortunately, current research literature has not always agreed on the overall benefits of pacing. In this paper, we present a model for the effectiveness of pacing. Our model demonstrates that for a given buffer size, as the number of concurrent flows are increased beyond a *Point of Inflection* (PoI), non-paced TCP outperforms paced TCP. We present a lower bound for the PoI and argue that increasing the number of concurrent flows beyond the PoI, increases inter-flow burstiness of paced packets and diminishes the effectiveness of pacing.**

I. Introduction and Motivation

Throughput and latency in the Internet are heavily influenced by the behaviour of TCP. Briefly stated, TCP is a window-based congestion control scheme: during each round-trip time (RTT), every TCP source transmits packets with total size equal to its congestion window. Most TCP variants, such as Tahoe, Reno, NewReno [10], BIC [27], and CUBIC [13], focus on the evolution of the congestion window size over time, and ignore the details of how packets are injected into the network in sub-RTT time scales. However, ignoring sub-RTT burstiness can produce bursty traffic on high bandwidth networks [28] such as data centres. This, in turn, produces more queueing delays, more packet losses, and lower throughput [2]. TCP pacing [28] addresses the problem by ensuring that bursts of packets do not cause contention in the router buffers. Specifically, TCP pacing evenly spaces the transmission of a window of packets over an entire RTT, so that data are not sent in a burst. This approach allows the sender to increase its sending rate without creating bottleneck queues [17].

Using TCP pacing is becoming more relevant in the context of data center networks, as the ever-increasing link speeds mean greater difficulty designing routers with buffer sizes equal to the bandwidth-delay product (BDP). As a result, many data center networks use switches with small buffers. Yet small-buffer switches and small RTTs in data center infrastructures create a disparity between the total capacity of the network and the capacity of individual queues, a disparity large enough to make TCP's congestion control scheme inefficient. In addition, the latency-sensitive, bursty traffic in distributed systems compels us to closely consider the impact of TCP pacing in data center environments. In such networks, an important performance criterion, besides the average latency, is the 99^{th} percentile tail latency; i.e., the completion time of the slowest flows. For example, in a search engine data center, a search query is distributed among thousands of servers; the median latency has no bearing on the result because the engine has to wait for the slowest flows to finish. In such environments, short term unfairness in TCP causes tail latency to grow, an issue that pacing may be able to address.

In this paper, we study the effectiveness of TCP pacing in inter-data center transactions, where RTTs are higher than 1ms and bounded by the geographical distance of data center sites. We confirm that by avoiding bursty packet drops, paced TCP achieves higher throughput than non-paced TCP. However, the benefits of using paced TCP diminish as we increase the number of concurrent connections beyond a certain threshold; if the number of flows goes beyond that point, non-paced TCP can perform better than paced TCP. We define the *Point of Inflection* (PoI) as the point in terms of number of concurrent flows where non-paced TCP out-performs paced TCP; we quantitatively determine its lower bound in terms of link capacity (C), RTT and buffer size (B), as $\frac{C \times RTT}{B}$.

Our results yield two new insights. First, it may not be appropriate to derive conclusions about pacing by studying a network with a fixed number of users and various buffer sizes. In fact, depending upon the BDP-to-Buffer ratio, the benefits of pacing varies. Second, increasing the number of flows produces inter-flow burstiness (bursts of packets belonging to different flows) which causes drop synchronization, leading to performance degradation. In previous studies, Kulik *et al.* [17] report throughput improvement over simulated satellite links and propose pacing over the entire lifetime of a TCP connection. However, the simulation study by Aggarwal *et al.* [2] argues that, contrary to intuition, pacing has a negative impact on performance. In Section IV, we determine whether our model can justify these this latter contradiction.

To validate our model, we perform several experiments in a test-bed. We find that while the number of concurrent flows is below the PoI bound, pacing offers improvements on link utilization, drop rate, average and 99^{th} percentile flow completion times. As the number of flows passes twice the PoI bound, however, these benefits are diminished

978-0-7695-5103-6/13 $26.00 © 2013 IEEE
DOI 10.1109/HOTI.2013.18

II. Background and Related Work

A data center is commonly defined as a site where multiple servers are co-located. However, cloud services may involve multiple geographically distributed data centers. These sets of sites are connected by dedicated links and sometimes used as complete replicas of the same service.[1] In this paper, we refer to data center traffic as workloads between multiple sites, with RTTs between 1 to hundreds of milliseconds bounded by the geographical distance of data center sites.

Unlike the long-lived and non-bursty streaming traffic on the Internet, the traffic patterns between data center nodes are bursty and mostly consist of short-sized flows because traffic in such networks belongs to distributed applications that perform scatter/gather type of communications, including MapReduce [9], Google File System (GFS) [11], Panasas [25], pNFS [21], and Hadoop [22]. For example, MapReduce uses Remote Procedure Call (RPC) protocol [23] for communication between services in distributed systems to request a service from a program located in a remote computer through network. Meanwhile, in GFS, a transfer service manages all data transfers, each with one master and several workers. The master distributes jobs to the workers; each job is divided into RPC flows with few mega bytes in size. The transfer service starts the flows and continue until the transfer is finished.

The bursty properties of TCP differ for the above cases and for conventional large-sized, streaming traffic. In streaming traffic, TCP by itself starts to *pace* its window due to the ACK-clocking dynamics [15], [18]. Specifically, the transmission of new packets is controlled by the stream of received ACKs and, hence, there are small increases in the window size. In a data center environment, each TCP flow is a long-lived connection between a client-server pair with many short-sized RPCs within it. In other words, a TCP connection serves several back-to-back RPC flows sharing the TCP parameters (such as congestion window). As a result, within a connection, each new RPC flow starts with the congestion window of the previously finished flow; thus, there is an obvious burst of up to a congestion window's worth of packets at the beginning of each RPC. In streaming, bursts can still happen but will be more subtle and less frequent. The bursty behaviour will only get worse when 10 $GigE$ network interface cards are deployed in data centers. Moreover, because data center networks have a single administrative domain and data center sites are connected to each other via dedicated links, the network administrator can control cross traffic which thereby affects the paced traffic.

The two major causes of high latency in data center networks are packet loss and TCP's slow recovery algorithm. TCP requires three duplicate ACKs to retransmit a lost packet, a threshold that is designed to avoid spurious retransmission due to packet reordering in the network. But this method is too conservative for data center environments where most losses are recovered by (conservative) timeouts. In such cases, TCP pacing may be a better option. TCP Pacing is a mechanism to combine pure rate control and TCP's window control to reduce the short-time-scale burstiness of TCP flows, without incurring significant delay penalties [2]. Zhang *et al.* [28] suggested TCP pacing to adjust the incorrect timing from acknowledgments (ACK) compression due to cross traffic. Other researchers have suggested using pacing when ACKs are not available to use for rate control, for example, to avoid TCP slow start at the beginning of a connection [19]. Similarly, pacing can be used to avoid burstiness in asymmetric networks caused by batching ACK packets [4]. While the theoretical benefits of pacing are clear, its behaviour in practical networks is not yet fully understood. There has been a wide set of simulation studies of TCP pacing with mixed and sometimes seemingly contradictory conclusions. More recently, researchers have observed performance improvements when deploying conventional pacing or a variation of it in various scenarios. Kamezawa *et al.* [16] propose a hardware approach to stabilize TCP on fat networks with large RTTs and large buffer routers and Beheshti *et al.* [5] demonstrate the necessity of pacing in routers with tiny buffers (10–15 packets).

Researchers have studied communication patterns in data centers in terms of high-fan-in, high-bandwidth synchronized TCP workloads, commonly known as the TCP incast problem [8], [24]. Such work is orthogonal in the sense that we are focusing on per flow burstiness which impacts packet drops and flow completion times. TCP pacing is not tied to any specific workload or TCP version: rather, pacing is about mitigating bursts, and bursts can occur regardless of TCP variant. The potential benefits of TCP pacing in data center environments include: (i) better link utilization on small-buffer routers, (ii) better short-term fairness among flows of similar RTTs by improving the worst flow latency, and (iii) smaller drop rates. Simply reducing the retransmission time-out (RTO) often makes no significant improvement since RTO activates a series of actions in the kernel, slowing the sending speed.

III. Modeling the effectiveness of pacing

Most TCP variants inject several packets back-to-back, wait for an entire RTT to acknowledge receipt, adjust the window size and repeat. TCP pacing, on the other hand, avoids sending bursts of packets by spreading packet injections over the course of one RTT, making the traffic smoother. Thus, TCP pacing is believed to be an effective way to mitigate the impact of traffic burstiness [28].

An interesting phenomenon occurs when we increase the number of flows in the network. Even though pacing removes the burstiness resulting from back-to-back packet injections in one flow, increasing the total number of flows creates another form of burstiness: packets belonging to different flows start creating bursts in the aggregated traffic. Such bursts worsen as we increase the number of flows such that, after a certain point, the impact of pacing goes away completely, and the performance begins to drop; performance becomes worse than when we have no pacing in the network.

[1] Replication is mostly used for reducing user latency and improving serving throughput [14].

Consider a network with a bottleneck link capacity C (packets/s) and buffer size B_{max} (packets) shared among N concurrent flows with the same round-trip time RTT (s).[2] Previous models of TCP mostly make the simplifying assumption that N and C are large enough so that the average window size, $\bar{W}$, does not depend on N [20], [26]. In our model, we explicitly avoid such simplification so that we can derive a bound on the number of flows at which the benefits of pacing diminishes.

We consider two identical systems, one with non-paced and one with paced TCP. At steady state, on average, each flow is sending $\bar{W}$ packets in every RTT. Further, assume RTT is divided into discrete time intervals of equal size δ.

We claim that for any given buffer size, the benefit of using pacing will be less pronounced as we increase N. In a perfectly paced world, every flow's packets are distributed evenly throughout the entire RTT. Hence, the probability of having one packet at time interval δ is a Bernoulli trial with $p = \frac{\bar{W} \times \delta}{RTT}$. Now, consider the aggregate of N paced flows and define random variable X as the total number of packet arrivals in interval δ. X is a binomially distributed random variable $X \sim B(N, p)$. Returning to our network, X represents the burst of inter-flow packets in δ time interval. As the number of concurrent flows (N) increases, the fair share of each flow decreases and so does $\bar{W}$ and, hence, p. However, the probability distribution of X becomes larger as N increases, causing an increase in burstiness.

Now consider the following two systems: the best case of non-paced traffic (BNP), and the worst case of paced traffic (WP). Both systems contain the sequence of bursts of packets to the bottleneck link during one RTT. The best case scenario of non-paced traffic happens when RTT is divided into equal intervals of $\Delta_1 = \frac{RTT}{N}$ and each flow sends $\bar{W}$ packets at the beginning of each interval, making the aggregate traffic a series of bursts spaced with Δ_1 intervals. Here, we are making a simplifying assumption that each non-paced flow transmits a burst size of average length $\bar{W}$ packets clustered together. This assumption is based on the clustering of packets in non-paced TCP [2] and as we show in Section IV-D is quite realistic.

In WP, the worst case of paced traffic, RTT is divided into equal intervals of $\Delta_2 = \frac{RTT}{\bar{W}}$ and each flow is sending one packet at the beginning of each interval. The packets from different flows make a burst of size N. Similar to BNP, we are making the realistic assumption that all paced flows are intermixed with each other. The worst case of paced traffic happens when packets of each flow are equally spaced through entire RTT with interval Δ_2.

Below we show that WP has better performance in terms of queue occupancy compared with BNP as long as:

$$N < \bar{W} = \frac{C \times RTT}{B_{max}} \tag{1}$$

Up to this point, paced TCP outperforms non-paced TCP; passing this point however, the benefits start to diminish. Since we are comparing the best and worst cases of the two scenarios here, Equation 1 proves the following lower bound on the PoI.

Theorem 1: In a network with N flows, bottleneck capacity C, and buffer size B_{max}, the lower bound on the Point of Inflection, the number of flows at which non-paced TCP starts to outperform paced TCP, is:

$$N^* \quad = \quad \Omega(\frac{C \times RTT}{B_{max}}) \tag{2}$$

Proof: See Appendix. ■

[2]The RTT here stands for the latency between data center sites due to their geographical distance and thus the homogeneous assumption is valid.

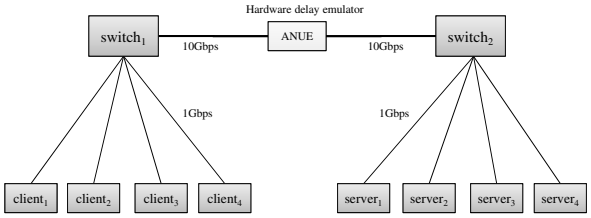


Fig. 1. Experiment's test-bed topology.

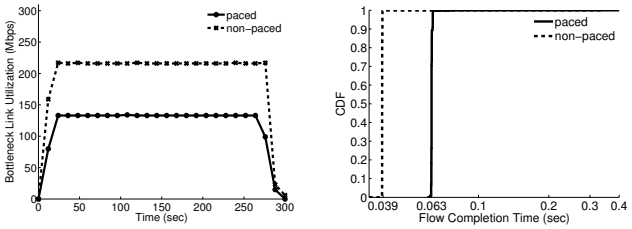
IV. Experiments

To validate our model, we perform several experiments using TCP pacing-enabled Linux kernels. We evaluate the impact of pacing on bottleneck link utilization, loss rate, drop synchronization, and average and 99^{th} percentile flow completion times. We also demonstrate the existence of the PoI and the accuracy of our model in estimating the lower bound of PoI. In what follows, Section IV-A explains the setup for the experiments, Section IV-B illustrates the basic advantages and disadvantages of pacing, and Section IV-C generalizes the previous section and presents the results of a more realistic model consisting of multiple flows.

A. Setup of Experiments

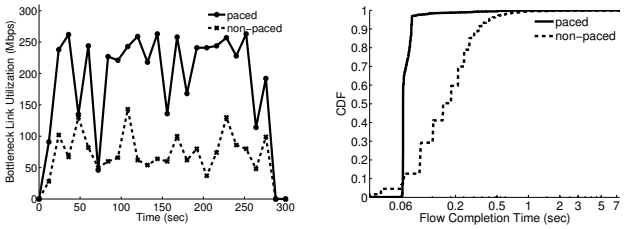
Figure 1 illustrates the topology used for our experiments. TCP flows travel between a set of senders and receivers through a single bottleneck link. The bottleneck link switches have configurable buffers and use FIFO scheduling and drop tail buffer management. To monitor the bottleneck link utilization and drop rate, we directly read the port statistics of the switches every 12 seconds. The default capacity of the bottleneck link is 10 $Gbps$ and the capacity of access links are 1 $Gpbs$. We use the ANUE hardware delay emulator [1] to emulate delay in the network. The ANUE receives data and stores them on a large buffer and transmits the stored data from another port after a specified delay time has passed. The direction of flows in this topology is from server hosts to client hosts. As a result, $switch_2$ is the congestion point.

We perform experiments using TCP CUBIC, NewReno, and BIC congestion control algorithms. We use a Netperf request size of $20B$ and a response size of 1 MB, 2 MB, or a combination of 100 B to 1 MB flow sizes. Throughout our experiments, we limit the bottleneck bandwidth to 1, 2, and



(a) (b)

Fig. 2. Base-case (i): Under-subscribed bottleneck. **(a)** Bottleneck link utilization. **(b)** CDF of flow completion times.



(a) (b)

Fig. 3. Base-case (ii): Over-subscribed bottleneck. **(a)** Bottleneck link utilization. **(b)** CDF of flow completion times.

3 $Gbps$ and use the delay emulator to emulate various ranges of RTTs, from 1 ms to 100 ms. Our results share the same conclusions and in the following sections we use 1 $Gbps$ bottleneck bandwidth, 1 MB flow sizes, 30 ms RTT, and TCP CUBIC congestion control algorithm,[3] unless otherwise stated. The run time for each experiment is 300 seconds. To obtain flow completion times, we use `tcpdump` to record packets on all hosts. We define Flow Completion Time (FCT) as the difference between the time that the TCP sender starts to send the first byte of each flow until the time it receives the ACK of the last byte.

Due to lack of space, in the following sections we report experiment results of 1 $Gbps$ bottleneck bandwidth, 1 MB flow sizes, 30 ms RTT, and TCP CUBIC congestion control algorithm, unless otherwise stated. Our results share the same conclusions and we refer the interested reader to our technical report [12] for the complete set of results.

B. Base-Case Experiment

We start with simple experiments with only one and two flows to demonstrate the potential effectiveness of TCP pacing in link utilization, packet drop, and flow completion time.

First, consider two simple cases: (i) having no congestion in the bottleneck link by limiting the number of concurrent TCP connections to only one from $server_1$ to $client_1$, and (ii) introducing congestion in the bottleneck link by adding a second concurrent connection between $server_2$ and $client_2$.

We limit the buffer size at both switches to 1% of the BDP; *i.e.* 64 KB. In case (i), the capacity of the access links and the bottleneck link are equal; thus, there will be no congestion at the bottleneck link. However, in case (ii), the ratio of access to core bandwidth is two, and the bottleneck link is congested. The two simple cases illustrate two intuitive laws of TCP pacing: (i) when there is no congestion, and hence, no loss at the bottleneck link, TCP pacing performs worse than non-paced TCP simply because there is no need to pace packets, and (ii) TCP pacing improves performance by mitigating loss events in the buffers when the bottleneck link is congested.

Under-subscribed bottleneck: Figure 2(a) compares the bottleneck link utilization of paced and non-paced cases through the entire 300 seconds of the experiment. Note that because there are no packet drops, non-paced TCP achieves 38% higher utilization than paced TCP. This is easily explained by looking at flow completion times, illustrated in Figure 2(b). In our setup, the transmission time of a 1 MB RPC is 8 ms. Nearly all flows finish within one RTT (39 ms in the figure) in non-paced TCP because after the initial slow start, the congestion window becomes fully open; the following RPC calls take advantage of this fully open congestion window and send the whole 1 MB in one burst. But because TCP pacing adds delay between packets, the paced flows need 2.1 RTTs (63 ms in the figure) to finish. Even though the congestion window is fully open, the last batch of packets is sent at the end of the RTT; the sender needs another RTT to receive the ACK packets and initiate the next RPC call.

Over-subscribed bottleneck: We introduce congestion in the network by adding another concurrent TCP connection between a new client-server pair: ($server_2$, $client_2$). Figure 3(a) compares paced and non-paced cases through the entire 300 seconds. With congestion in the network, paced TCP achieves higher utilization than non-paced TCP. Non-paced TCP is experiencing around 0.5% of packet drops at the bottleneck queue. This phenomenon is highly undesirable in data center networks as TCP loss recovery is not efficient for small latency-sensitive transfers. The paced flows have almost zero drops. As shown in Figure 3(b), their flow completion time is still mostly two RTTs, where-as non-paced flows have considerably larger average and 99^{th} percentile flow completion times.

C. Multiple Flows

In this section, we consider a more realistic case whereby many flows are sharing the bottleneck link. Our results confirm the model proposed in Section III; for any given buffer size, as we increase the number of concurrent connections beyond the PoI, non-paced TCP out-performs paced TCP. In Section III, we quantitatively determined a lower bound for PoI (N^*) in terms of link capacity (C), RTT and buffer size (B), as $\frac{C \times RTT}{B}$.

Figures 4, 5, and 6 illustrate the above claim when the buffer size is 64, 128 and 256 KB, respectively. We increase the total number of concurrent flows from 4 to 100 using all four client-server pairs in Figure 1 by distributing the load

[3]TCP CUBIC is the default congestion control algorithm in Linux kernels since version 2.6.19.

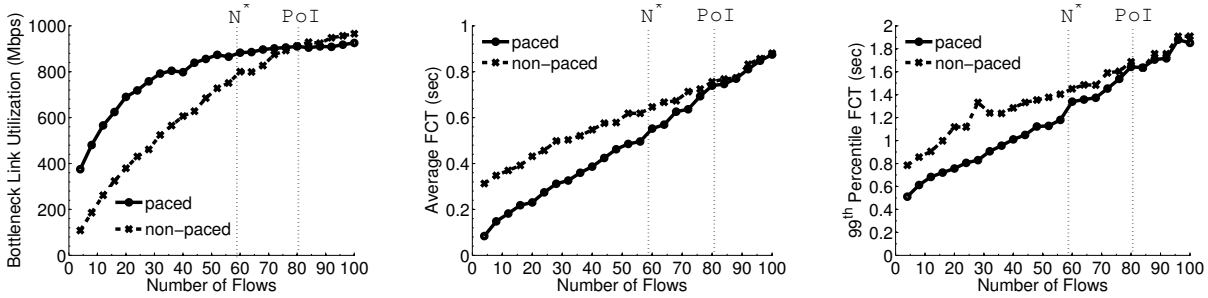


Fig. 4. Studying the effect of number of flows sharing the bottleneck link when the buffer size is 1.7% of BDP (64 KB) and the lower bound on inflection point (N^*) is 58 flows. **(a)** Bottleneck link utilization. **(b)** Average flow completion times. **(c)** 99^{th} percentile flow completion times.

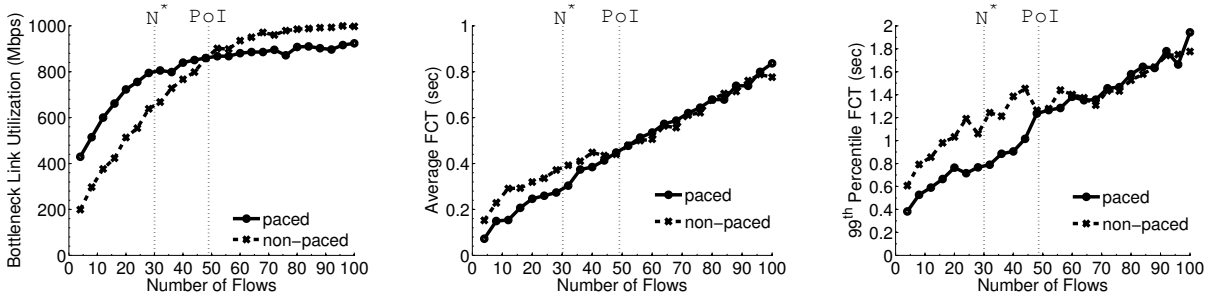


Fig. 5. Studying the effect of number of flows sharing the bottleneck link when the buffer size is 3.4% of BDP (128 KB) and the lower bound on inflection point (N^*) is 29 flows. **(a)** Bottleneck link utilization. **(b)** Average flow completion times. **(c)** 99^{th} percentile flow completion times.

evenly across hosts. The capacity of the access links and the bottleneck link are both equal to 1 $Gbps$. As a result, the bottleneck is congested and as explained in Section IV-B, we expect paced TCP to have higher link utilization and lower flow completion times. However, as discussed in Section III, this also depends on the BDP-to-Buffer ratio.

Figure 4(a) depicts the bottleneck link utilization versus the number of concurrent flows sharing the bottleneck link. Paced traffic achieves higher utilization than non-paced traffic when the number of flows is below 80 (the PoI). Pacing also has better average and 99^{th} percentile of RPC completion times, as shown in Figures 4(c) and (d), respectively.

The benefits diminish as we increase the number of flows beyond 80. In this setup the BDP-to-Buffer ratio (N^*) is 58 flows which is the lower bound on taking advantage of pacing, as proposed in Section III. Note that we might be able to achieve some improvement in performance with pacing when the number of flows is greater than this lower bound, as shown in Figure 4, but the benefits have already started to diminish.

Repeating the same experiment with different buffer sizes, as illustrated in Figures 5 and 6, makes it clear that for every given buffer size, as the number of flows increases, the benefits of pacing become less pronounced. Figure 5 illustrates the same experiment as Figure 4 but here the available buffer size has been doubled. Interestingly, the PoI in Figure 5 (48) is almost half of the PoI in Figure 4 (80). In Figure 6, when the buffer size is doubled, the PoI is reduced to 19 flows.

All three experiments are performed with buffer sizes of 1.7%, 3.4% and 6.8% of BDP, but measuring the drop rate at the bottleneck link shows that 6.8% of BDP is enough to have no packet drops even with non-paced traffic. Hence, they constitute a spectrum of tiny, small and large buffers in the context of this experiment.

In the experiments shown in Figures 4 and 5, we see that paced traffic always has a lower drop rate than non-paced traffic, even after the PoI. However, as we will discuss in details in Section IV-D, the synchronization of drops in paced TCP causes many flows to experience a loss event, leading to a dramatic degradation in performance.

An interesting question is the accuracy of the lower bound derived in Section III. In other words, for various buffer sizes, what happens if we set the number of flows to be exactly $N^* = \frac{C \times RTT}{B_{max}}$? As we discussed in our model, N^* is the lower bound on the PoI. We therefore, expect to see paced and non-paced TCP perform about the same when the number of flows is exactly N^*. Figure 7 confirms this by plotting the bottleneck link utilization and flow completion times for various buffer sizes when the number of flows are chosen according to Equation 1. As expected, paced and non-paced cases perform similarly. Thus, the lower bound N^* is a valid

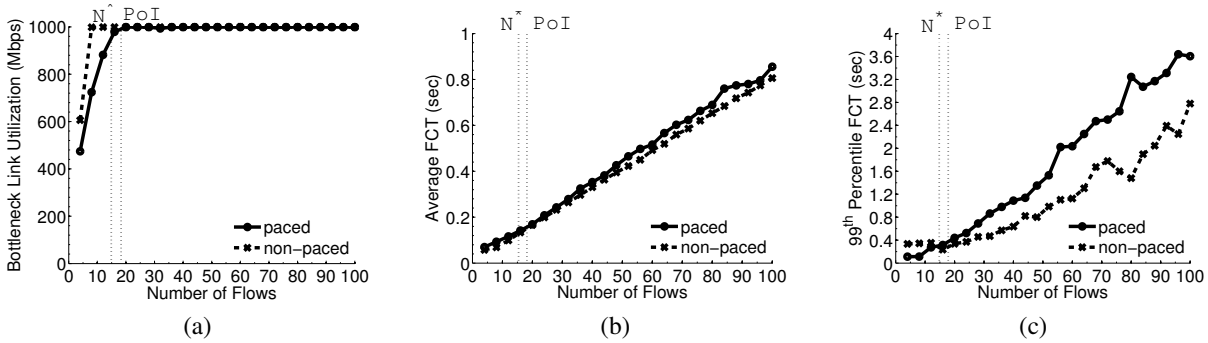


Fig. 6. Studying the effect of number of flows sharing the bottleneck link when the buffer size is 6.8% of BDP (256 KB) and the lower bound on inflection point (N^*) is 14 flows. **(a)** Bottleneck link utilization. **(b)** Average flow completion times. **(c)** 99^{th} percentile flow completion times.

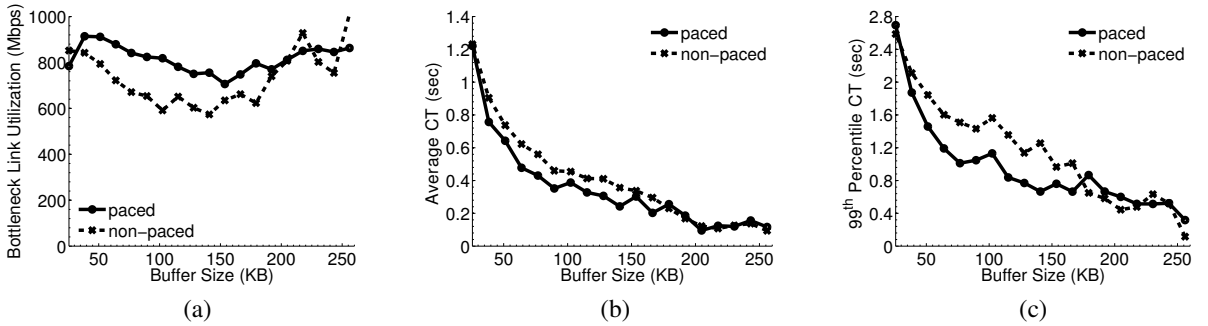


Fig. 7. Studying the accuracy of N^* lower bound on the PoI when the buffer size varies. **(a)** Bottleneck link utilization. **(b)** Average flow completion times. **(c)** 99^{th} percentile flow completion times.

threshold for the PoI. Moreover, throughout the experiments presented here and in our technical report [12], we observe that the PoI is always less than twice the N^* bound. In other words, in our experiments, the following inequality always holds: $N^* \leq PoI \leq 2 \times N^*$.

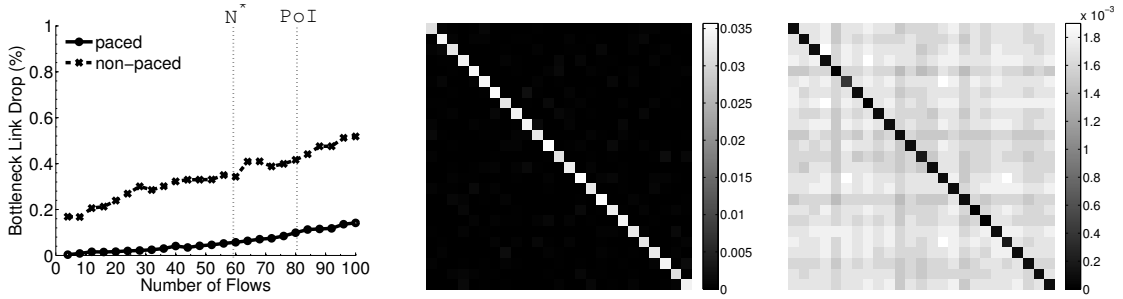
Our BDP-to-Buffer ratio rule can justify the seemingly contradictory conclusions in previous works. For example, Aagarwal *et al.* [2] study a network with 50 long-lived flows with BDP of 1250 packets and buffer size of 312 packets. Our results suggest $PoI = 8$ flows is the point of inflection in such a network and increasing the number of flows to 50 diminishes pacing's effectiveness. On the other hand, Kulik *et al.* [17] study networks with BDP of 91 packets and buffer sizes of 100 and 10 packets with a single TCP flow which lies within our bound.

D. Drop Synchronization

In this section, we study the impact of inter-flow burstiness on drop synchronization. As mentioned in Section IV-C, in our experiments, paced traffic always experiences lower drop rate compared with non-paced traffic. For example, Figure 8(a) compares the drop rate of paced TCP with non-paced TCP in the experiment illustrated in Figure 4. Even after the PoI, paced TCP continues to have a lower drop rate compared with non-paced TCP. However, as we will show in this section, the synchronization of drops in paced TCP causes many of the flows to experience the loss event; hence, a dramatic degradation in performance. The negative effect of drop synchronization in the slow start phase of paced TCP has been observed and reported in previous studies of TCP pacing [2]. In the following, we demonstrate that drop synchronization in paced TCP is not tied to slow start phase and is a function of number of flows in the network: increasing the number of concurrent flows sharing the bottleneck increases the inter-flow burstiness, and as a result, increases the chance of many flows experiencing the drop event.

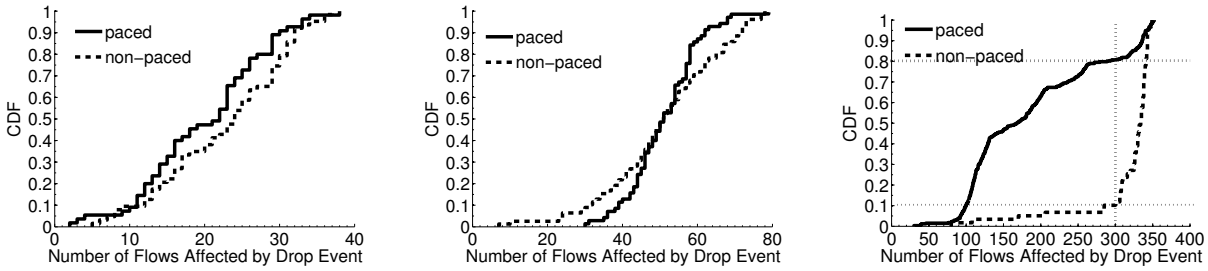
To provide the intuition for the drop synchronization phenomenon and to verify the simplifying assumptions in Section III, we first measure the probability of packet clustering in paced and non-paced cases. Figures 8(b) and (c) illustrate the clustering matrix for one of the hosts in our topology when there are 100 flows in the bottleneck link (25 flows from each host).[4] Each figure shows a 25×25 colored matrix where the color of each element (i, j) in the matrix corresponds to the probability that a packet from flow i is followed by a packet from flow j. In Figure 8(b), the diagonal of the matrix has the lightest color (largest probability), and hence clearly shows that in the non-paced case, it is highly likely that packets from each flow are clustered to each other. On the other hand, the

[4]This corresponds to the very last data point in Figure 8(a).



(a) (b) (c)

Fig. 8. (a) Bottleneck link's drop rate. (b) The clustering probability matrix for 25 non-paced flows: the color of element (i, j) corresponds to the probability that packets from flow i are followed by packets from flow j. **(c)** The clustering probability matrix for 25 paced flows: the color of element (i, j) corresponds to the probability that packets from flow i are followed by packets from flow j.



(a) (b) (c)

Fig. 9. CDF of flows that are affected by a drop event reported by the NetFPGA router. **(a)** Total number of flows is 48. **(b)** Total number of flows is 96 flows. **(c)** Total number of flows is 384.

diameter of the matrix in Figure 8(c) has the darkest color (lowest probability) which shows that in paced case it is more likely that packets from different flows are intermixed with each other.

In order to measure the effect of drop synchronization, we use a NetFPGA router to count the number of flows that are affected by drop events at the bottleneck router. The NetFPGA router is configured with the "event capturing module" of the NetFPGA router design [5] which supports instrumenting the router's output queues. [5] We modify the event capturing module to report the flow ID of flows that experience drop events. Figure 9(a), (b), and (c) illustrate the CDF of flows that are affected by a packet drop event with paced and non-paced TCP when the total number of flows in the bottleneck link is 48, 96, and 384, respectively. In the case of 48 flows, illustrated in Figure 9(a), paced and non-paced TCP behave roughly similar to each other, with paced TCP having slightly larger probability of drop synchronization. Increasing the number of flows to 96 (Figure 9(b)) increases the chance of drop synchronization in paced flows. If the number of flows are further increased to 384, the synchronization effect becomes more dominant. Figure 9(c) illustrates that 300 flows (78% of total number of flows) are affected by 80% of drop events in paced TCP; but, the same number of flows are only affected by 10% of drop events in non-paced TCP.

[5]When a packet arrives, departs, or is dropped at an output queue, the current clock of the NetFPGA which has an 8 ns granularity, is recorded. Multiple events are then collected in a single event packet which is periodically sent out a specified router port.

V. Conclusions and Future Trends for TCP Pacing

This paper studies TCP pacing in data center networks and presents a unifying model for the general effectiveness of pacing. We show that for a given buffer size, as the number of concurrent flows are increased beyond a certain bound, the benefits of using paced TCP will start to diminish. We provide a lower bound for this point of inflection in terms of link capacity (C), RTT and buffer size (B), as $\Omega(\frac{C \times RTT}{B})$. Moreover, we demonstrate that increasing the number of flows produces inter-flow burstiness (bursts of packets belonging to different flows) which causes drop synchronization and hence performance degradation. We validate our model using a novel and practical implementation of paced TCP in the Linux kernel and perform several experiments in a test-bed.

In the networking literature, TCP pacing usually refers to per-flow smoothing of congestion window size of packets within one RTT. In this paper, we showed that having many paced flows introduces inter-flow burstiness and drop synchronization which hurts the effectiveness of pacing. We propose applying per-host TCP pacing on top of per-flow pacing to reduce the inter-flow bursts and drop events. The objective of per-host pacing, or more precisely per-egress port pacing, is

to smooth the aggregate traffic leaving an egress port. To that end, and by knowing the rate R_i of each flow exiting an egress port of a host, a per-host pacer applies an inter-transmission gap of $\delta = \frac{MTU}{\Sigma_i R_i}$ between all packets. As a result, per-host pacing smooths the transmission of aggregate of flows leaving an egress port and alleviates the impact of inter-flow bursts. Essentially, at steady state, this results in having almost zero packet drops at the bottleneck link. We leave the study of per-egress port pacing to interested researchers in the field.

ACKNOWLEDGEMENTS

We would like to thank Glen Anderson, Nandita Dukkipati, Jerry Chu, Lawrence Brakmo, Joe Ethier, Steve Padgett, and Ashish Naik from Google Inc. for helping us with some of our experiments. We would also like to thank Yashar Ghiassi-Farrokhfal, Phillipa Gill and Mukarram bin Tariq for useful discussions and comments on the early versions of this paper. This work was supported by Google Inc. as well as CGS grant from Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] ANUE systems. http://www.anuesystems.com/index.shtml.
[2] A. Aggarwal, S. Savage, and T. Anderson. Understanding the performance of TCP pacing. In *IEEE Infocom*, pages 1157–1165, 2000.
[3] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). *In SIGCOMM*, August 2010.
[4] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz. The effects of asymmetry on TCP performance. In *MobiCom*, pages 77–89, 1997.
[5] N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, and G. Salmon. Experimental study of router buffer sizing. In *IMC*, pages 197–210, 2008.
[6] Y. Cai, Y. S. Hanay, and T. Wolf. Practical packet pacing in small-buffer networks. In *IEEE ICC*, pages 1–6, 2009.
[7] K. Chen, P. Huang, C. Huang, and C. Lei. The impact of network variabilities on tcp clocking schemes. In *INFOCOM*, pages 1–6, 2006.
[8] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP incast throughput collapse in datacenter networks. In *WREN*, pages 73–82, 2009.
[9] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI*, pages 10–10, 2004.
[10] S. Floyd and T. Henderson. RFC2582: The NewReno modification to TCP's fast recovery algorithm, 1999.
[11] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. *SIGOPS OSR*, 37(5):29–43, 2003.
[12] M. Ghobadi, G. Anderson, Y. Ganjali, J. Chu, L. Brakmo, and N. Dukkipati. TCP pacing in data center networks. Technical report, TR10-SN-UT-04-10-00, University of Toronto, April 2010.
[13] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS OSR*, 42(5):64–74, 2008.
[14] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.
[15] K. Jacobsson, L. L. H. Andrew, A. Tang, K. H. Johansson, and S. H. Low. ACK-clocking dynamics: Modeling the interaction between windows and the network. In *INFOCOM*, pages 2146–2152, 2008.
[16] H. Kamezawa, M. Nakamura, J. Tamatsukuri, N. Aoshima, M. Inaba, and K. Hiraki. Inter-layer coordination for parallel TCP streams on long fat pipe networks. In *SC*, page 24, 2004.
[17] J. Kulik, R. Coulter, D. Rockwell, and C. Partridge. A simulation study of paced TCP. Technical report, BBN Technical Memorandum No. 1218, 1999.
[18] J. C. Mogul. Observing TCP dynamics in real networks. In *SIGCOMM*, pages 305–317, 1992.
[19] V. N. Padmanabhan and R. H. Katz. TCP fast start: A technique for speeding up web transfers. In *IEEE Globecorn Internet Mini-Conference*, pages 41–46, 1998.
[20] G. Raina and D. Wischik. Buffer sizes for large multiplexers: TCP queueing theory and instability analysis. In *Next Generation Internet Networks*, pages 173–180, 2005.
[21] S. Shepler, M. Eisler, and D. Noveck. RFC3530: Network File System version 4 protocol.
[22] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.
[23] R. Srinivasan. RFC1831: RPC: Remote Procedure Call protocol, 1995.
[24] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *SIGCOMM*, pages 303–314, 2009.
[25] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou. Scalable performance of the Panasas parallel file system. In *FAST*, pages 17–33, 2008.
[26] D. Wischik. Buffer sizing theory for bursty tcp flows. In *IZS*, pages 98–101, 2006.
[27] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (BIC) for fast long-distance networks, 2004.
[28] L. Zhang, S. Shenker, and D. D. Clark. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. In *ACM SIGCOMM CCR*, pages 133–147, 1991.

APPENDIX

Proof of Theorem 1:

In both systems, at steady state, each flow is sending $\bar{W}$ packets at every RTT and thus we have $C \times RTT = N \times \bar{W}$. The burst size in BNP and WP is $\bar{W}$ and N, respectively.

In BNP, when the burst size of non-paced traffic is less than or equal to the available buffering capacity, there is no need to pace packets within the congestion window. Thus:

$$\begin{aligned} \bar{W} &\leq B_{max} \\ \frac{C \times RTT}{N} &\leq B_{max} \\ \frac{C \times RTT}{B_{max}} &\leq N \end{aligned} \tag{3}$$

Similarly, in WP, pacing is outperformed by a non-paced system if:

$$\begin{aligned} N &\geq B_{max} \\ \frac{C \times RTT}{N} &\geq B_{max} \\ \frac{C \times RTT}{B_{max}} &\geq \bar{W} \end{aligned} \tag{4}$$

In other words, for $N = \bar{W} = \frac{C \times RTT}{B_{max}}$, WP and BNP become similar systems. Thus, to take advantage of pacing one should choose the number of flows so that $N < \bar{W} = \frac{C \times RTT}{B_{max}}$.