Networking 2013 1569696437

Edge versus Host Pacing of TCP Traffic in Small Buffer Networks

Hassan Habibi Gharakheili
School of Electrical Engineering and Telecommunications
UNSW, Sydney, Australia
Email: h.habibi@student.unsw.edu.au

Arun Vishwanath
Centre for Energy-Efficient Telecommunications
University of Melbourne, Australia
Email: arun.v@unimelb.edu.au

Vijay Sivaraman
School of Electrical Engineering and Telecommunications
UNSW, Sydney, Australia
Email: vijay@unsw.edu.au

***Abstract*—As packet switching speeds scale to Terabits-per-second and beyond, power considerations are increasingly forcing core router manufacturers to adopt all-optical and hybrid opto-electronic single-chip switching solutions. Such routers will have small buffers, typically in the range of a few tens of Kilobytes, causing potentially increased packet loss, with adverse impact on end-to-end TCP performance. We recently proposed and analysed the benefits of pacing traffic at the network edge for open-loop real-time traffic in a small buffer network. However, no detailed study of the efficacy of edge pacing on closed-loop TCP performance has been undertaken for such a network.**

In this paper, we consider two pacing methods - TCP pacing at the end-hosts, and traffic pacing by the network edge - in the context of small buffer networks, and undertake a comprehensive comparison. Our contributions are three-fold: First, we show via extensive simulations that under most scenarios (considering bottleneck and non-bottleneck core links, low-speed and high-speed access links, long- and short-lived TCP flows, and different variants of TCP) edge pacing performs as well or better than host pacing in terms of link utilisation (TCP throughputs) and average per-flow goodputs. Second, we provide analytical insight into the setting of the edge pacing delay parameter, showing how the efficacy of pacing relates to bottleneck buffer size. Third, we discuss incremental deployment of pacing, highlighting that unlike host pacing that requires adoption by a critical mass of users, edge pacing can be deployed relatively easily under service provider control to facilitate rapid migration to core networks with small buffers.

I. Introduction

As Internet traffic continues its inexorable growth, core routers are struggling to keep pace with the required switching capacity. Router scaling is primarily limited by power density – a typical rack today with a throughput of a Terabit-per-second consumes tens of KiloWatts, and at current trends, scaling its capacity to Petabits-per-second would require hundreds of KiloWatts of power, alongside complex cooling mechanisms. To sustain capacity growth, router manufacturers are therefore increasingly looking to photonics, including all-optical packet switching solutions and integrated single-chip systems employing hybrid optics and electronics. In order to perform energy-efficient high-speed packet forwarding, such architectures necessarily sacrifice many non-critical functionalities, among them buffering of packets during periods of congestion. Recent research studies on such architectures have argued, based on theory, simulation, and experimentation, that core router buffer size can safely be reduced from Gigabytes down to Megabytes [1] or Kilobytes [2], and can even be nearly eliminated [3], though with some loss in performance. We refer the reader to our survey article [4] for a comprehensive discussion on the router buffer sizing debate.

When router buffers in the network core are very small (sub-50 KB), contention and congestion at the output link can lead to high packet loss, significantly degrading end-to-end traffic performance. We have shown in [5] that real-time traffic streams can experience poor quality, and in [6] that TCP flows can have reduced throughput. Several mechanisms have been proposed for mitigating this problem, such as using wavelength conversion [7] in the core to alleviate contentions, using packet-level forward-error-correction (FEC) at edge nodes to recover from core loss [3], and traffic pacing at the edge prior to injection into the core [5], [8], [9]. While all these methods have their relative merits, in this paper we focus on pacing, since it is low-cost (compared to wavelength conversion), is a well-known concept (studied under various names such as rate-limiting, shaping, smoothing, etc.), and is yet relatively unexplored in the context of small buffer networks.

Traffic can be paced in various parts of the network: by end-hosts as part of their TCP stack (host pacing), by the access link connecting the user to the network (link pacing), or by the edge node that connects into the core network (edge pacing). *Host pacing* (also known as TCP pacing) modifies the end-user client TCP stack to spread the transmission of packets from the TCP window over the round-trip-time (RTT) of the connection. Many researchers have studied host pacing over the past decade [10], [11], [12], and the general belief is that host pacing can, under most circumstances, improve overall TCP throughput. However, deploying host pacing has been stymied by the fact that the network operator does not have control over user devices to enforce pacing, and hosts that pace their TCP transmissions can be unfairly penalised over hosts that do not [11].

Link pacing relies on the access link being of much lower capacity than links deeper in the network, ensuring that packets belonging to any single flow are spaced apart when they arrive at the core link. This has been leveraged by works such as [2] to argue that neither can a single flow contribute bursty traffic to the core node, nor are many flows likely

to synchronise to create bursts, and hence loss is contained. Though this argument applies to typical home users, entities such as enterprises, universities, and data centers are often serviced with high-speed links capable of generating bursty traffic that does not fit this assumption, necessitating explicit mechanisms (at the host or edge) to reduce burstiness.

Edge pacing relies on explicit smoothing of traffic by edge nodes prior to injection into the small buffer core network. In [5] we proposed a method that adjusts traffic release rate to maximise smoothness, subject to a given upper bound on packet delay. We proved the optimality of our scheme, analysed its burstiness and loss performance, and evaluated its impact for open-loop real-time traffic, though not for closed-loop TCP. A similar (though sub-optimal) pacing method was proposed in [8] to vary the edge traffic release rate based on queue backlog. However, the impact of edge pacing on TCP performance was only cursorily studied, and no appropriate guidelines on parameter settings were provided.

Our goal in this paper is to undertake a comprehensive comparison between host and edge pacing in the context of small buffer core networks, by evaluating their impact on end-to-end TCP performance. We seek to gain insights into the network and traffic characteristics that influence their efficacy, the parameter settings that maximise their benefits, and deployment strategy that make them practical in real networks. Our specific contributions are:

- We show using extensive simulations of various scenarios, considering small-buffered bottleneck and non-bottleneck links, low-speed and high-speed access links, short- and long-lived flows, different number of flows, and different variants of TCP, that edge pacing achieves as good or better performance than host pacing in terms of link throughput and per-flow goodput.
- We develop an analytical model that sheds light into the selection of the edge pacing delay parameter that maximises TCP throughput for different bottleneck link buffer sizes.
- We argue that the benefits of edge pacing can be easily realised under tight operator control in real networks, unlike host pacing that requires a critical mass of uptake by end-users for it to be effective.

Our intention is to show network operators that from a performance, configuration and deployment point-of-view, edge pacing presents an attractive alternative to host pacing as a mechanism for enabling scalable and energy-efficient core networks having small-buffers in the near future.

The rest of this paper is organised as follows: Section II gives requisite background on traffic pacing studies. In Section III we present comprehensive simulation studies comparing the performance of host and edge pacing, and in Section IV we develop analytical insights into appropriate parameter settings. Section V discusses the deployment strategy for pacing, and the paper is concluded in Section VI.

II. Background and Related Work

It is well-known that TCP traffic is bursty at short-time scales [13] because of its self-clocking mechanism and queueing of packets at the bottleneck link. Bursty traffic is largely undesirable since it causes large queueing delays, higher packet loss, and degradation in end-to-end throughput. As a result, several researchers have proposed to pace TCP at the end-hosts, an idea initially suggested by [14], to reduce burstiness. A comprehensive simulation study to evaluate the benefits of end-host TCP (Reno) pacing is undertaken by [11], who argue that pacing can result in lower throughput and higher latencies for most realistic network settings. Since packets across different TCP flows are evenly spaced, the flows can become synchronised and experience simultaneous losses at the bottleneck link, leading to lower throughput than unpaced flows. They also point out that paced flows perform poorly when coexisting with unpaced flows in the network.

However, as noted in a more recent study [10], there is no consensus on whether end-hosts should pace TCP. The paper evaluates via analysis and simulations the impact of pacing not just TCP Reno, but also newer protocols such as New Reno, SACK and FACK. The authors conclude that it is indeed beneficial to pace TCP at the end-hosts, and that the performance when all flows pace is better than when no flows pace. Further, when the fraction of paced flows exceeds a critical value, both the paced and unpaced flows gain in performance. The experimental study using a high-speed wide area network [15] showed that the overall throughput of parallel TCP transfers improves substantially when pacing is employed, while [16] found that pacing can improve the aggregate TCP throughput of multiple Reno and FACK flows in large bandwidth-delay product networks by 20%.

It must be mentioned that the above studies assume end-host pacing and consider bottleneck link with large buffers (i.e. at least an order of magnitude more than our study). This study differs in two ways: (a) We consider pacing traffic at the network edge and compare it to TCP pacing by end-hosts, and (b) We consider small buffers (sub-50 KB) at the bottleneck link, motivated by the move towards all-optical and hybrid opto-electronic switching solutions. Our earlier work [5] developed an edge pacing method (details described next) for reducing traffic burstiness at the edge of the small buffer core network, proved its optimality, and evaluated its performance via native simulation for open-loop real-time traffic. Parallel to our work, [8] also developed a similar (though sub-optimal) edge pacing mechanism termed Queue Length Based Pacing (QLBP). However, their method uses three parameters (compared to one in our case), and does not undertake a comprehensive study of TCP performance.

In contrast, our work in this paper is the first to undertake a thorough evaluation of closed-loop TCP performance (by implementation in *ns-2*) in the presence of edge and host pacing in a small buffer network, under a variety of network settings using both aggregate throughput and average per-flow goodput as metrics.

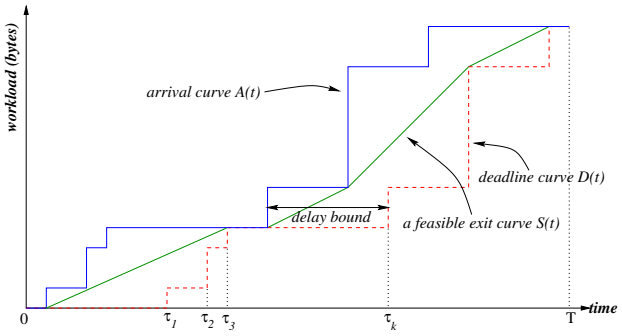


Fig. 1. Arrival, deadline and exit curves for an example workload

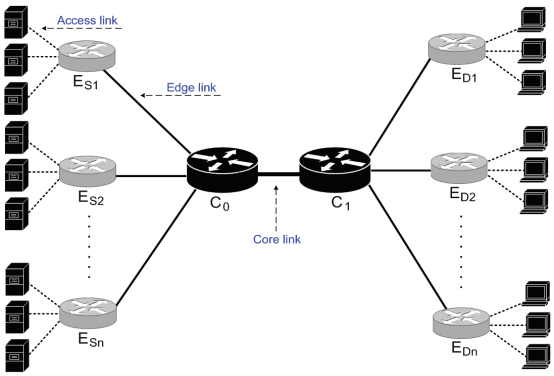


Fig. 2. *ns-2* network topology

A. Edge pacing mechanism

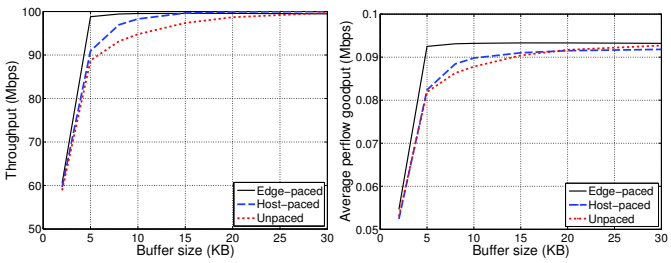
The edge pacing mechanism used in this study is based on the optimal algorithm we developed in [5]. Our pacer, unlike a shaper that releases traffic at a given rate, accepts arbitrary traffic with given delay constraints, and releases traffic that is "smoothest" (i.e. has lowest maximum rate and rate variance) subject to the time-constraints of the traffic. Fig. 1 depicts an example traffic arrival curve $A(t)$ (i.e. the cumulative arriving workload in units of bytes), from which the deadline curve $D(t)$ (i.e. the cumulative workload that has to be served so as not to violate any deadlines) is derived, based on configured parameter d corresponding to the maximum delay that the pacer is allowed to introduce. A feasible exit curve $S(t)$ must lie in the region bounded above by the arrival curve $A(t)$, and below by the deadline curve $D(t)$. Amongst all feasible exit curves, we have shown that the one which corresponds to the smoothest output traffic is the shortest path between the origin $(0, 0)$ and $(T, D(T))$, as shown in Fig. 1. In our earlier work we showed that an online implementation of optimal pacer would compute the convex hull of the deadline curve, and use the corresponding instantaneous slope as the rate at which to release traffic. We also showed that the convex hull can be computed in $O(1)$ amortised time, and is amenable for high-speed hardware implementation. The pacing delay bound d is a critical parameter that determines the window of time over which pacing is effective – when $d = 0$, pacing is in effect disabled, since packets cannot be held back by the pacer. As the delay bound d increases, the traffic becomes increasingly smooth. Further details of the pacing mechanism, and an analysis of its impact on traffic burstiness and loss performance for open-loop real-time traffic, can be found in [5].

III. Efficacy of Edge Pacing for TCP Traffic

The above mentioned pacer was implemented in version 2.33 of the *ns-2* network simulator. We created a new link type, by extending the *drop-tail* link, and incorporated the computation of the convex hull as per the $O(1)$ amortised time algorithm. The patch for end-host TCP pacing was obtained from [17] and runs in *ns-2* version 2.28.

Using the implementation in *ns-2*, we conducted extensive simulations to evaluate the effectiveness of pacing TCP traffic at the network edge, and compared with the performance of non-paced and host-paced flows using aggregate TCP throughput and average per-flow goodput as metrics. We use goodput as a metric since it has been argued to be the most important measure for end-users [18], who want their transactions to complete as fast as possible. All simulations in this section use an edge pacer delay bound of 10ms; we will justify this choice via analysis in the next section.

A. Small buffer link as the bottleneck



(a) Aggregate TCP throughput (b) Average per-flow goodput

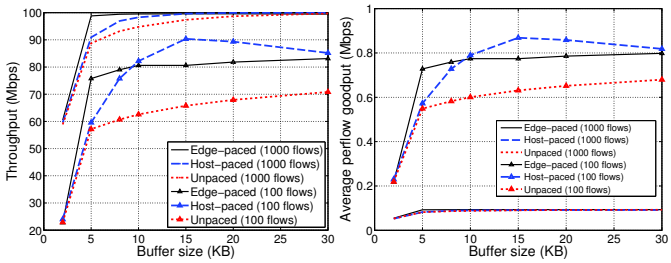
Fig. 3. TCP performance with small buffer link as the bottleneck

Our simulations were conducted on the single link dumbbell topology shown in Fig. 2. Ten ingress edge links (E_{S1}-E_{S10}) feed traffic into the core link C_0-C_1, with each edge link in turn fed by hundred access links. Each end-host has one TCP (Reno) agent, and the network therefore simulates 1000 long-lived TCP flows (short-lived flows and different TCP versions are discussed below). Similarly the TCP flows are sinked by the 1000 end-hosts on the right, which are connected to ten egress edge links (E_{D1}-E_{D10}) . The propagation delays on the access and edge links are uniformly distributed between $[1, 5]$ ms and $[5, 15]$ ms respectively, while the core link C_0-C_1 has delay 100 ms. RTTs therefore vary between $[224, 280]$ ms. The access link speeds are uniformly distributed in $[8, 12]$ Mbps, all edge links operate at 100 Mbps, and the core link also at 100 Mbps. For these simulation settings it can be seen that the core link is the bottleneck. FIFO queue with drop-tail queue management is employed at C_0, and the queue size is varied in terms of KB. Data and ACK packet sizes are 1000 and 40 Bytes respectively. The start time of the TCP flows is uniformly distributed in the interval $[0, 10]$ sec and the

simulation is run for 400 sec. Data in the interval $[100, 400]$ sec is used in all our calculations so as to capture the steady-state behaviour of the network.

Fig. 3(a) shows the aggregate TCP throughput as a function of core link buffer size. When buffers are very small (i.e 2-3 KB), packet loss rates at the core link were found to be in excess of 15% for the three scenarios shown in the figure. The benefit of pacing packets (at the host or the edge) is thus outweighed by the high loss rates, and the aggregate TCP throughput is no better than when all flows are unpaced. On the other hand, the efficacy of pacing packets at the edges is pronounced in the small buffer regime (i.e. 5-15 KB), reflected in the aggregate TCP throughput shown in Fig. 3(a) as well as the average per-flow goodput in Fig. 3(b). At 5 KB worth of buffering, the per-flow goodput for host and unpaced flows is ≈ 82 Kbps, while edge pacing achieves 93 Kbps. Edge pacing therefore outperforms host pacing by over 13%. As core buffers get larger (i.e. > 20 KB), the utilisation of the link C_0-C_1 is near-100%, and therefore there is no room for pacing to improve TCP performance, suggesting that edge pacing is particularly beneficial in the region of 5-15 KB buffers.

B. Number of TCP flows



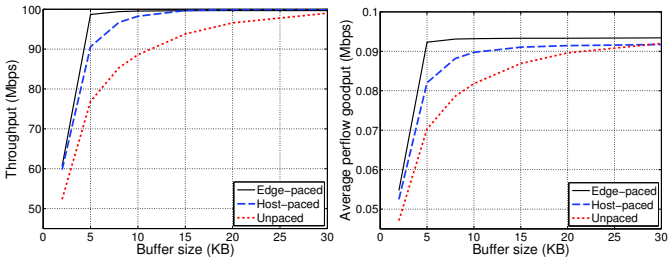
(a) Aggregate TCP throughput (b) Average per-flow goodput

Fig. 4. TCP performance with 100 and 1000 flows

We now study the efficacy of the pacer for varying number of TCP flows. We use the same setup as before, but alter the number of access links (10, 50, 100) feeding into the edge, to simulate 100, 500 and 1000 flows respectively. The resulting aggregate TCP throughput and average per-flow goodput are shown in Fig. 4 (plots corresponding to 500 flows closely follow that of 1000 flows, not plotted for the sake of clarity).

In contrast to the previous case of 1000 flows where edge pacing consistently outperformed host pacing over the entire buffer size range, when the number of flows is small (100) and buffer sizes are in a certain region (10-30 KB), we find that host pacing gives better performance than edge pacing. We believe this is because of the following: burstiness at the bottleneck buffers can arise in two ways – (1) an individual flow itself can generate bursty traffic and contribute to increased loss, or (2) packets from multiple sources might arrive simultaneously to cause loss. When the number of flows is small, the burstiness of an individual flow is greater as its TCP window expands to a larger value, and this contributes more to loss than the simultaneous arrival of packets from multiple flows. This can be seen to be trivially true in the case of only one flow in the network. Conversely when the number of flows is large, loss is more likely to happen due to simultaneous arrival of packets from several flows rather than due to many packets from one flow being in the buffer. Host pacing is more effective at reducing source burstiness (scenario 1) because it spaces traffic over a larger window (i.e. a RTT), whereas edge pacing deals better with the latter (scenario 2) since it can space the release of packets arriving simultaneously from multiple flows. This explains why pacing at the hosts is more beneficial than pacing at the edge when the number of flows is small. In practice however, core links typically have tens of thousands of TCP flows traversing through them, suggesting that it is better to pace the aggregate (at the edges) rather than the individual (at the host) for improved TCP performance.

C. High-speed access links



(a) Aggregate TCP throughput (b) Average per-flow goodput

Fig. 5. TCP performance with high-speed access links

We now investigate the merits of pacing TCP traffic in the presence of high-speed access links arising from data centres, enterprise and university networks, etc. We use the setup discussed above for 1000 flows, the difference being that access links now operate at 100 Mbps. The core link still remains the bottleneck. The bottom curve in Fig. 5(a) shows that at 5 KB of buffering, unpaced flows obtain an aggregate throughput of 77 Mbps, host pacing (middle curve) increases the throughput to 90 Mbps (better by 17%), and edge pacing (top curve) further pushes the throughput to 98 Mbps (improvement of 27% compared to unpaced flows), highlighting the efficacy of pacing traffic at the edge. Edge pacing obtains higher TCP throughput than host pacing in the small buffer regime (5-15 KB). Fig. 5(b) depicts the average per-flow goodput for the 1000 TCP flows and demonstrates that edge pacing is extremely effective. To obtain 90 Kbps goodput (90% of 0.1 Mbps, the ideal goodput) the bottom curve indicates that unpaced flows require 20 KB of buffering. Pacing flows at the host (middle curve) halves the buffering requirements to 10 KB, while edge pacing (top curve) achieves 90 Kbps with just under 5 KB buffers (half of that for host pacing, and a fourth of that for unpaced flows). These results highlight the efficacy of edge pacing for use with high-speed access links.

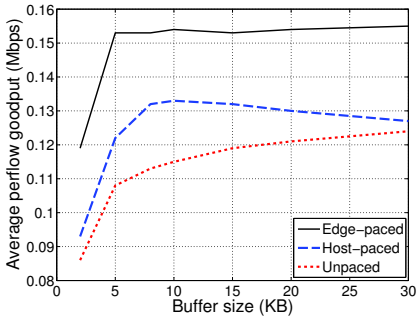
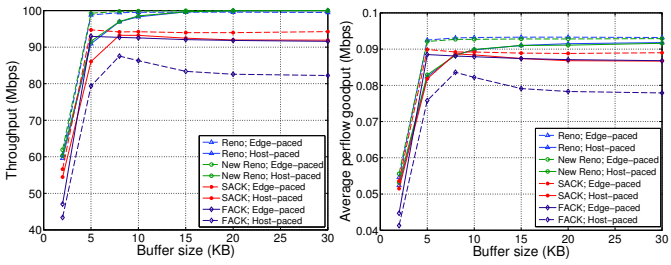


Fig. 6. Average per-flow goodput with short-lived TCP flows

D. Short-lived TCP flows

Our study thus far only considered long-lived TCP flows. We now consider short-lived TCP flows (also known as mice), wherein the number of active TCP flows is time-varying. Measurement studies in the Internet core show that a large number of TCP flows (e.g. HTTP requests) are short-lived. They spend most of their time in the slow-start phase and generate more bursty traffic than long-lived flows, that are often in the congestion avoidance mode. To incorporate such realistic TCP traffic we simulate the closed-loop flow arrival model described in [19], operating as follows. A given number of users perform successive file transfers to their respective destination nodes. The size of the file to be transferred follows a Pareto distribution with mean 100 KB and shape parameter 1.2. These chosen values are representative of Internet traffic, and comparable with measurement data. After each file transfer, the user transitions into an idle ("thinking period") or off state. The duration of the "thinking period" is exponentially distributed with mean 1 sec. We implemented this model in *ns-2* and repeated our simulations on the dumbbell topology with 1000 short-lived flows. Fig. 6 shows that pacing TCP at the edge can improve the average per-flow goodput of short-lived flows substantially, peaking at 155 Kbps with 10 KB of buffering, which is nearly 17% larger than the goodput obtained by pacing TCP at the end-hosts (133 Kbps). These results with short-lived flows demonstrate that edge pacing is very effective in combating short time-scale burstiness (typical of short-lived TCP flows).

E. Different versions of TCP

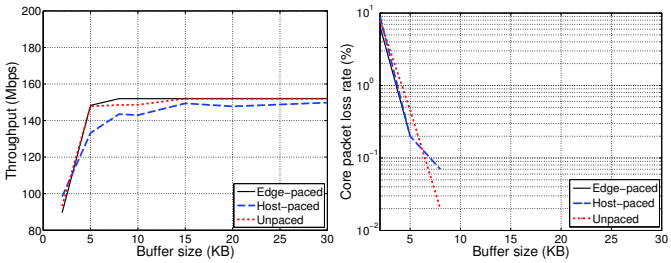


(a) Aggregate TCP throughput (b) Average per-flow goodput

Fig. 7. Performance with different TCP variants

We compared the performance of edge and host pacing with three additional variants of TCP (New Reno, SACK and FACK). Simulations in Section III-A were repeated with each of these TCP versions. Overall, we observed that for all the above variants of TCP, edge pacing offers better performance than host pacing (typically by more than 10% in the region 5-15 KB) in terms of aggregate throughput as well as average per-flow goodput, as depicted in Fig. 7.

F. Small buffer link not the bottleneck



(a) Aggregate TCP throughput (b) Packet loss rate at the core link

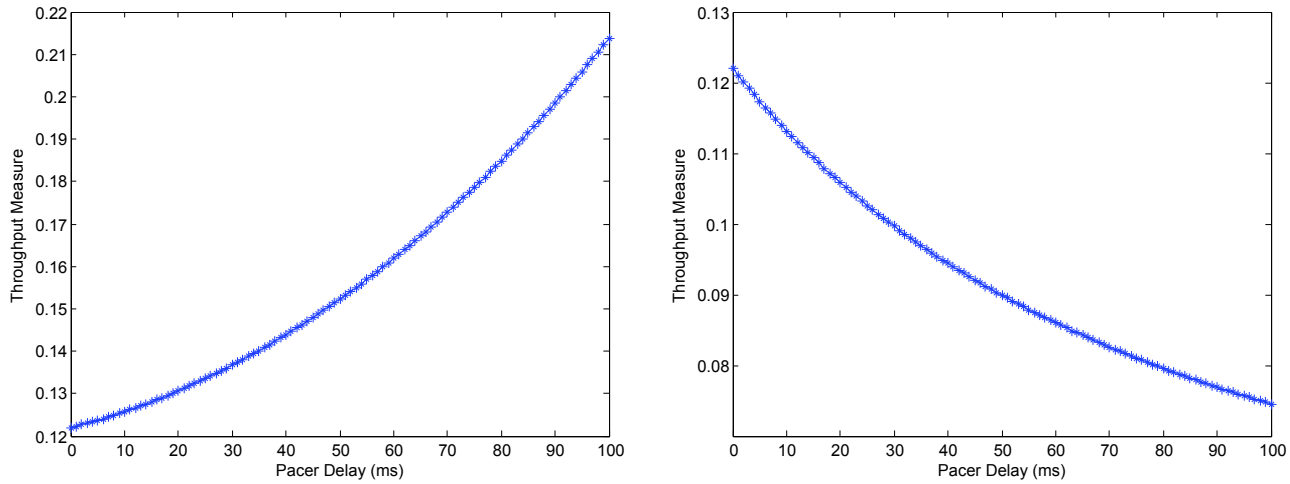
Fig. 8. TCP performance with small buffer link as non-bottleneck

All the previous scenarios considered the small buffer core link as the bottleneck link. We analysed the impact of pacing TCP traffic when the core link is not the bottleneck link. To this end, we set the core and edge link rates to 200 Mbps and 40 Mbps, and access link rates are uniformly distributed in $[1, 2]$ Mbps respectively. 10 access links feed into each edge link, with 10 edge links in turn feeding into the core. In all, the network simulates 100 long-lived TCP flows. Since the access network is the bottleneck, and 10 edge links feed into the core, it is evident that the core link does not require more than 10 KB of buffering to guarantee zero packet loss. This can be seen from Fig. 8(b), which plots the loss rate (on log-scale) as a function of buffer size. The aggregate throughput curve in Fig. 8(a) shows that pacing (both host and edge) achieves approximately the same performance as unpaced, indicating that TCP throughput is not sensitive to pacing when the small buffer link is not the bottleneck.

The above results illustrate, under various network settings, that pacing traffic at the edge of a small buffer network is extremely effective in obtaining high TCP throughput and per-flow goodputs, and can play an important role in the design of future generation optical core networks with limited buffering capability.

IV. Analysing the Impact of Edge Pacer Delay

In this section we seek to develop insights into the impact of pacing on TCP performance. Modeling TCP performance is notoriously difficult due to its control feedback loops, and indeed existing models of host pacing often resort to (excessively conservative) worst-case approximations to bound performance. We will resort to several simplifications and approximations, with a view towards getting insight into the



(a) Low load/Small buffer

(b) High load/Large buffer

Fig. 9. Throughput measure as a function of pacer delay from our analytical model for (a) Low load (small buffer), and (b) High load (large buffer)

shape of curves relating edge pacing delay parameter d with TCP throughput, rather than their exact numbers.

We begin with the relatively well-known fact that throughput of a TCP flow is inversely proportional to its RTT as well as to the square root of the loss L it experiences:

$$T \propto \frac{1}{RTT\sqrt{L}} \tag{1}$$

Pacing traffic at the network edge smoothens the aggregate TCP traffic, reducing loss at bottleneck buffers. However, pacing holds packets back in the pacer queue, which increases the mean end-to-end RTT. In what follows we attempt to quantify these two opposing forces, and show that the force that dominates to determine the appropriate choice of pacing delay parameter d depends on factors such as network buffer size and traffic flow characteristics.

Quantifying the impact of edge pacing on RTT is relatively easy: pacing with delay bound d adds roughly $d/2$ delay to each packet on average in each direction, increasing mean RTT to $RTT_0 + d$, where RTT_0 is the round-trip-time without pacing.

Quantifying loss at a buffer fed by several (paced) TCP sources is however non-trivial. A worst-case assumption that all TCP sources synchronise their bursts (to yield a giant saw-tooth) is unrealistic (especially when thousands of TCP sources share the link) and excessively conservative. We instead resort to the observation (made in our earlier work [20] and by others e.g., [21], [22]) that the aggregated traffic from a large number of TCP flows sharing a small buffer (up to 50 Kilobytes) is approximately Poisson. We have shown buffer occupancy traces in [20] to substantiate that large bottleneck buffers cause TCP flows to synchronise, whereas small buffers break this synchrony, and aggregation therefore allows application of the central limit theorem to allow Poisson approximation. Hence, in what follows we assume that the aggregate TCP traffic is Poisson-like with a certain (yet to be determined) rate λ.

When Poisson traffic of rate λ is fed into an edge pacer with delay parameter d, the egress traffic has burstiness (ratio of standard deviation to mean rate) given by [5]:

$$\beta = 1/\sqrt{2\lambda d} \tag{2}$$

Further, the loss rate, derived using a bufferless fluid approximation, is obtained from the Chernoff bound as [5]:

$$L \leq (\lambda e^{1-\lambda})^{2d} \tag{3}$$

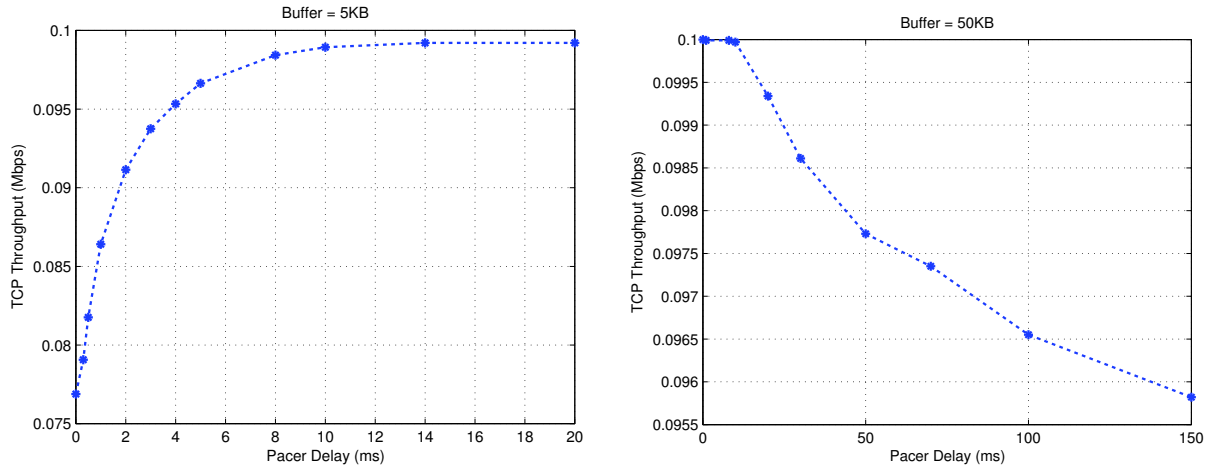
This shows that loss falls monotonically as the pacer delay d is increased. Moreover, since the above bound is derived under a fluid approximation, it holds irrespective of the number of edge nodes that pace traffic prior to aggregation at the core node buffers, as long as the aggregate rate is λ.

With the above expressions for RTT and L, we can rewrite the throughput of a TCP flow from (1) as:

$$T \propto \frac{1}{(RTT_0 + d)(\lambda e^{1-\lambda})^d} \tag{4}$$

We plot this in Fig. 9 for two cases: Fig. 9(a) considers relatively light load and plots the above throughput measure for a base round-trip-time RTT_0 fixed at 200 ms, and the pacer delay d is varied from 0 to 100 ms. We see that the curve is monotonically increasing, suggesting that larger pacing delay values are preferable, since the benefits of loss reduction from smoothing outweigh the penalty due to increased RTT. In Fig. 9(b) we consider the case of heavy load and plot the above throughput measure for the same base round-trip-time RTT_0 of 200 ms, and the pacer delay d varied from 0 to 100 ms. In this case, we find that the per-flow TCP throughput falls monotonically with pacing delay, suggesting that when loads are higher, larger pacing delays are detrimental as the effect of increased RTT outweighs the benefits of reduced loss from smoothing.

Having argued that the net effect of pacing delay on TCP throughput depends on the offered load λ, we argue that the offered load directly depends on traffic characteristics



(a) Low load/Small buffer

(b) High load/Large buffer

Fig. 10. Throughput measure as a function of pacer delay from simulations for (a) Low load (small buffer), and (b) High load (large buffer)

and bottleneck link buffers. When many TCP flows share bottleneck link buffers, we have shown in [6] (and several other researchers have corroborated [23]) that the empty buffer probability $1-\rho$ falls exponentially with buffer size, and hence the offered load is $\rho = 1 - e^{-B/B^*}$ where B represents bottleneck buffer size and B^* is a constant (with same units as B) dependent on system parameters such as link capacities, number of flows and their durations, round-trip-times, etc (we found B^* to be in the range 2-10 KB). Thus, when buffer size is small (say 5 KB), the offered load is lower $\rho \approx 90\%$, and as buffer size increases (to say 50 KB), offered load increases to over $\rho \approx 99.9\%$.

To validate that smaller bottleneck buffers favour a higher pacing delay, we ran simulations using the same topology as in the previous sections - 1000 TCP flows share a bottleneck core link of 100 Mbps capacity. We set the core link buffers to 5 KB, and plot in Fig. 10(a) the per-flow TCP throughput. It shows that TCP throughput increases with pacing delay, as predicted by our analysis in Fig. 9(a). We now set the core link buffers to 50 KB, and plot in Fig. 10(b) the per-flow TCP throughput obtained from simulation. We find that in this case TCP throughput falls with pacing delay, as predicted by our analysis in Fig. 9(b), since the offered load with larger buffers is higher and the benefits of loss reduction are outweighed by increase in RTT.

The above analysis provides valuable insight into the relationship between pacing delays and TCP performance, and we do not claim to be able to accurately quantify TCP throughput. Indeed, though our analysis and simulation both show a monotonic rise in TCP throughput with pacing delay for low load (small buffers), the analysis curve in Fig. 9(a) is convex while the simulation curve in Fig. 10(a) is concave – this is because our analysis assumed a fixed load λ, whereas when the pacing delay is increased and loss reduces, TCP reacts by increasing its offered load. This increase in load can offset the loss reduction (it can be seen that the TCP throughput curve saturates in simulation when the pacing delay reaches 10 ms), whereas we do not capture this effect in our analysis (which is why the TCP throughput in our analysis continues to increase). Capturing these feedback effects precisely in a finite buffer system is notoriously hard, and is beyond the scope of the current paper. What we have established is that pacing delays need to be tuned to network and traffic conditions, and our observations from simulation show that pacing with larger delays is increasingly beneficial as the bottleneck buffers become smaller, especially when they fall below 10 KB.

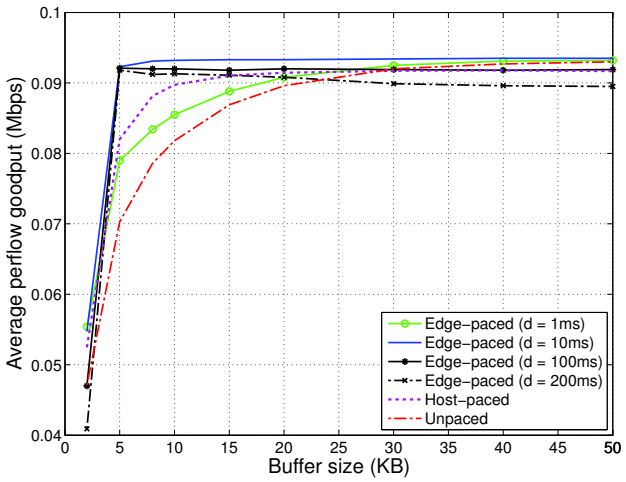
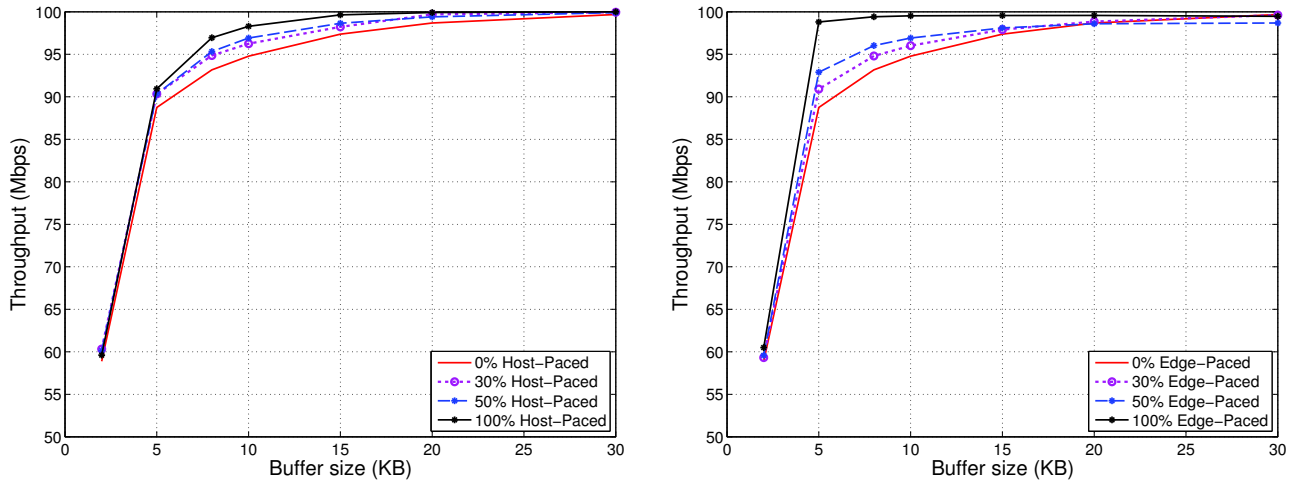


Fig. 11. Per-flow TCP goodput for various pacing delay values

In Fig. 11 we show the per-flow TCP goodput observed in simulation as a function of buffer size for various pacing delay values $d = 1, 10, 100, 200$ ms. It is observed that a small pacing delay $d = 1$ ms is relatively ineffective at small buffer sizes, while a large pacing delay such as $d = 100$ or 200 ms is detrimental (i.e. reduced TCP goodput) as buffer sizes become larger. Throughout our simulations we found that $d = 10$ ms was a good compromise that works well across the entire range of buffer sizes for all scenarios considered, and hence our simulation studies presented in other sections of this paper have used this delay value.



(a) Host pacing

(b) Edge pacing

Fig. 12. Aggregate bottleneck link throughput versus buffer size for varying fractions of pacing deployment at (a) Hosts and (b) Edge nodes

V. Practical Deployment of Pacing

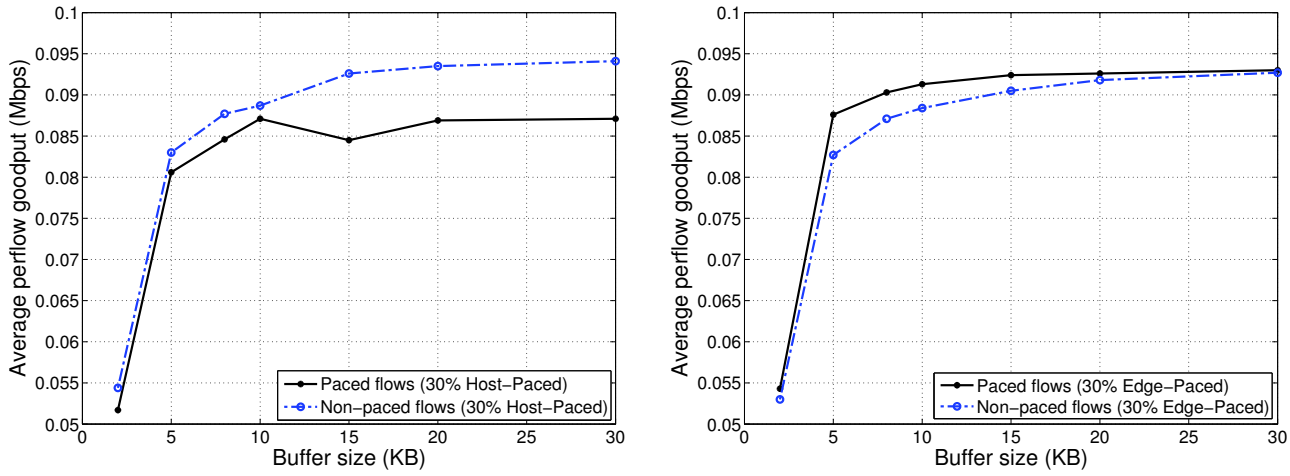
The previous sections have shown that in small buffer networks edge pacing is as effective (or more) as host pacing in improving link throughput and per-flow goodput. We now argue that edge pacing is also more easily deployed in an incremental way into operational networks. The first question we address is the benefit of partial deployment of pacing, namely when only a fraction of hosts (in the case of host pacing) or edge nodes (in the case of edge pacing) perform pacing. We conducted simulations under various scenarios with fractional deployment of pacing, and measured the impact on aggregate throughput improvement on the bottleneck link. Fig. 12 depicts results from one setting in which 1000 TCP flows multiplex at the bottleneck link, and shows the aggregate throughput when pacing is 0%, 30%, 50%, and 100% deployed at host and edge nodes respectively. Comparing host pacing in Fig. 12(a) with edge pacing in Fig. 12(b), we note that in both cases throughput rises gradually as the fraction of hosts/edges that perform pacing increases, and therefore it would seem the benefits of pacing can be realised incrementally with progressive deployment.

Though fractional deployment of pacing leads to *overall* throughput improvement for both host and edge pacing, the way these benefits are shared is radically different in the two cases. To illustrate this, consider the same scenario as before (i.e. 1000 flows share the bottleneck link), and say pacing is 30% deployed (namely, 300 out of 1000 flows perform TCP pacing in the case of host pacing and 3 out of 10 edge nodes perform pacing in the edge pacing case). In Fig. 13 we compare the average per-flow goodput of paced versus unpaced flows. Fig. 13(a) shows that with host pacing, flows that pace their TCP traffic actually obtain *worse* goodput (by as much as 10%) than flows that do not pace their traffic. This phenomenon has been identified in prior studies [11], [10] which have shown that TCP pacing is effective only if employed by a critical mass of users. Early adopters of host pacing can therefore obtain worse performance than their non-pacing peers, and this creates a substantial disincentive for users to adopt host pacing. By contrast, Fig. 13(b) shows that with the same fraction of flows being paced via edge pacing, this problem does not arise, and paced flows experience better performance than unpaced ones. This allows network operators to focus their initial deployment of edge pacing at sites connecting their most critical customers, confident that performance for these customers will not degrade (as in the case of host pacing).

Apart from the performance issues, there are also major logistic differences between deploying host and edge pacing. Host pacing requires changes to the TCP/IP protocol stack in the end-user client devices. Given the myriad devices in use today (PCs, laptops, tablets, smart-phones, Internet-enabled TVs, etc.) and their heterogeneous operating systems (Windows, Linux, iOS, Android, etc.), this is a daunting task. Further, the kernel update required to incorporate pacing at the host would need to be explicitly done by each user, which requires motivation and skill, and is virtually impossible to achieve at scale. Moreover, operating system vendors are reluctant to incorporate pacing in the standard kernel distribution for fear that initial adopters will get degraded performance. These factors have stymied deployment of TCP pacing over the last decade. By contrast, edge pacing has no such issues as it can be easily deployed since it is entirely under operator control. We have shown in our previous work [5] that our optimal edge pacing algorithm is amenable for hardware implementation at very high speeds, and operators can choose to employ it incrementally or simultaneously around their small buffer core network.

VI. Conclusions

Energy density concerns in modern high-speed routers are driving the trend towards photonic integrated switching platforms [24] with reduced buffering capability. In networks with



(a) Host pacing

(b) Edge pacing

Fig. 13. Per-flow goodput versus bottleneck buffer size for paced and unpaced flows with fractional deployment of pacing at (a) Hosts and (b) Edge nodes

such small buffers, end-to-end performance of TCP can be improved by traffic smoothing. In this paper, we compare two mechanisms of pacing – at the edge and the host – and undertake a comprehensive quantification of TCP throughput and per-flow goodputs in various scenarios comprising bottleneck and non-bottleneck links, short- and long-lived flows, low- and high-capacity access links, different number of TCP flows and various TCP variants. We showed that edge pacing performs as good, if not better, than host pacing. Via analysis and simulations we have provided insights into choosing the delay parameter of the edge pacer. We argued that unlike host pacing, there is a clear and safe path towards incremental deployment of edge pacing in an operational network. We offer edge pacing as an attractive solution for enhancing TCP performance in emerging all-optical or hybrid-optical core networks with small buffers.

REFERENCES

[1] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," in *Proc. ACM SIGCOMM 2004*, USA, Aug-Sep 2004.

[2] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, "Routers with Very Small Buffers," in *Proc. IEEE INFOCOM*, Spain, Apr 2006.

[3] A. Vishwanath, V. Sivaraman, M. Thottan, and C. Dovrolis, "Enabling a Bufferless Core Network using Edge-to-Edge Packet-Level FEC," in *Proc. IEEE INFOCOM*, USA, 2010.

[4] A. Vishwanath, V. Sivaraman, and M. Thottan, "Perspectives on Router Buffer Sizing: Recent Results and Open Problems," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 2, pp. 34–39, Apr 2009.

[5] V. Sivaraman, H. Elgindy, D. Moreland, and D. Ostry, "Packet Pacing in Small Buffer Optical Packet Switched Networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, pp. 1066–1079, 2009.

[6] A. Vishwanath, V. Sivaraman, and G. N. Rouskas, "Anomalous Loss Performance for Mixed Real-Time and TCP Traffic in Routers with Very Small Buffers," *IEEE/ACM Transactions on Networking*, vol. 19, no. 4, pp. 933–946, Aug 2011.

[7] E. M. Wong et al., "Towards a Bufferless Optical Internet," *IEEE/OSA Journal of Lightwave Technology*, vol. 27, no. 4, pp. 2817–2833, Jul 2009.

[8] Y. Cai, T. Wolf, and W. Gong, "Delaying Transmissions in Data Communication Networks to Improve Transport-Layer Performance," *IEEE Journal on Selected Areas in Communications*, vol. 29(5), pp. 916–927, 2011.

[9] B. Zhao, A. Vishwanath, and V. Sivaraman, "Performance of High-Speed TCP Applications in Networks with Very Small Buffers," in *IEEE Advanced Networks and Telecommunication Systems (ANTS)*, India, 2007.

[10] D. X. Wei, P. Cao, and S. H. Low. (2006) TCP Pacing Revisited. [Online]. Available: http://www.cs.caltech.edu/weixl/research/summary/infocom2006.pdf

[11] A. Aggarwal, S. Savage, and T. E. Anderson, "Understanding the Performance of TCP Pacing," in *Proc. IEEE INFOCOM*, Israel, 2000.

[12] C. Caini and R. Firrincieli, "Packet Spreading Techniques to Avoid Bursty Traffic in Long RTT TCP Connections," in *Proc. IEEE VTC Spring*, Italy, 2004.

[13] H. Jiang and C. Dovrolis, "Why is the Internet Traffic Bursty in Short Time Scales?" in *Proc. ACM SIGMETRICS*, Canada, 2005.

[14] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," in *Proc. ACM SIGCOMM*, Switzerland, 1991.

[15] H. Kamezawa et al., "Inter-Layer Coordination for Parallel TCP Streams on Long Fat Pipe Networks," in *IEEE/ACM Supercomputing*, USA, 2004.

[16] J. Kulik et al., "A Simulation Study of Paced TCP," *Tech. Rep. BBN Technical Memorandum No. 1218*, 1999.

[17] D. X. Wei. (2006) A TCP Pacing Implementation for NS2. [Online]. Available: http://netlab.caltech.edu/projects/ns2tcplinux/ns2pacing/index.html

[18] N. Dukkipati and N. McKeown, "Why Flow-Completion Time is the Right Metric for Congestion Control," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 59–62, 2006.

[19] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Closed Versus Open: A Cautionary Tale," in *Proc. USENIX NSDI*, USA, 2006.

[20] A. Vishwanath, V. Sivaraman, and D. Ostry, "How Poisson is TCP Traffic at Short Time-Scales in a Small Buffer Core Network?" in *Proc. IEEE Advanced Networks and Telecommunication Systems (ANTS)*, India, 2009.

[21] A. Lakshmikantha, C. Beck, and R. Srikant, "Impact of File Arrivals and Departures on Buffer Sizing in Core Routers," *IEEE/ACM Transactions on Networking*, vol. 19, no. 2, pp. 347–358, 2011.

[22] D.Wischik, "Buffer Sizing Theory for Bursty (TCP) Flows," in *Proc. 2006 International Zurich Seminar on Communications*, Switzerland, 2006.

[23] L. Andrew, T. Cui, J. Sun, M. Zukerman, K. Ho, and S. Chan, "Buffer Sizing for Nonhomogeneous TCP Sources," *IEEE Communications Letters*, vol. 9, no. 6, pp. 567–569, Jun 2005.

[24] I. Keslassy, S.-T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown, "Scaling Internet Routers Using Optics," in *Proc. ACM SIGCOMM*, Germany, 2003.