Comparing Edge and Host Traffic Pacing in Small Buffer Networks ☆

Hassan Habibi Gharakheili[a,1], Arun Vishwanath[b,2], Vijay Sivaraman[a,1]

[a] *School of Electrical Engineering and Telecommunications, UNSW, Sydney, Australia*
[b] *IBM Research, Melbourne, Australia*

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 demonstrate the benefits of pacing in practical scenarios multiplexing both TCP and real-time traffic, and 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.

Keywords: Small buffer networks, edge pacing, optical packet switched networks

1. 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 [2] or Kilobytes [3], and can even be nearly eliminated [4], though with some loss in performance. We refer the reader to our survey article [5] for a comprehensive discussion on the 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

☆ This submission is an extended and improved version of our paper presented at the IFIP Networking 2013 conference [1].

Email addresses: h.habibi@unsw.edu.au (Hassan Habibi Gharakheili), arvishwa@au.ibm.com (Arun Vishwanath), vijay@unsw.edu.au (Vijay Sivaraman)

[1] H. Habibi Gharakheili and V. Sivaraman are with the School of Electrical Engineering and Telecommunications, The University of New South Wales, Sydney, Australia.

[2] A. Vishwanath is with IBM Research, Melbourne, Australia.

degrading end-to-end traffic performance. We have shown in [6] that real-time traffic streams can experience poor quality, and in [7] that TCP flows can have reduced throughput. Several mechanisms have been proposed for mitigating this problem, such as using wavelength conversion [8] in the core to alleviate contentions, feedback-based rate control to proactively prevent network congestion in optical burst switching networks [9, 10], packet-level forward-error-correction (FEC) at edge nodes to recover from core loss [4], and traffic pacing at the edge prior to injection into the core [6, 11, 12, 13]. 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 [14, 15, 16], 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 [15].

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 [3] 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 [6] 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 [11, 12] 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 present evidence that edge pacing is beneficial in practical networks that multiplex both TCP and real-time traffic, and 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 2 gives requisite background on traffic pacing studies. In Section 3 we present comprehensive simulation studies comparing the performance of host and edge pacing, and in Section 4 we develop analytical insights

into appropriate parameter settings. Section 5 shows that edge pacing benefits both TCP and real-time traffic, Section 6 discusses the deployment strategy for pacing, and the paper is concluded in Section 7.

2. Background and Related Work

It is well-known that TCP traffic is bursty at short-time scales [17] 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 [18], to reduce burstiness. A comprehensive simulation study to evaluate the benefits of end-host TCP (Reno) pacing is undertaken by [15], 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 [14], 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 [19] showed that the overall throughput of parallel TCP transfers improves substantially when pacing is employed, while [20] 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 [6] 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, [11, 12] also developed a similar (though sub-optimal) edge pacing mechanism termed Queue Length Based Pacing (QLBP). However, their method uses three parameters (which can be more difficult to tune compared to just one parameter, the delay bound, in our case), and moreover does not undertake a comprehensive study of TCP performance when combined with real-time UDP traffic.

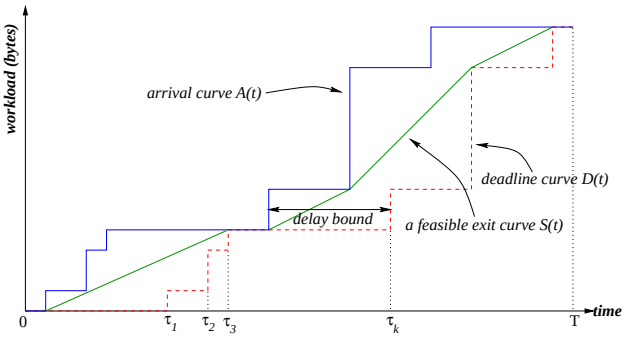


Figure 1: Arrival, deadline and exit curves for an example workload

In contrast, our work in this paper is the first to undertake a thorough evaluation of performance for both closed-loop TCP and real-time traffic (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. We will show in subsequent sections that edge pacing performs as good (if not better) than host pacing, and improves TCP performance significantly in a small buffer network.

2.1. Edge pacing mechanism

The edge pacing mechanism used in this study is based on the optimal algorithm we developed in [6]. 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 re-

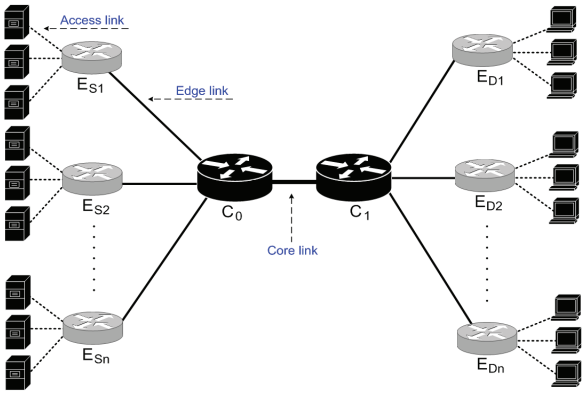
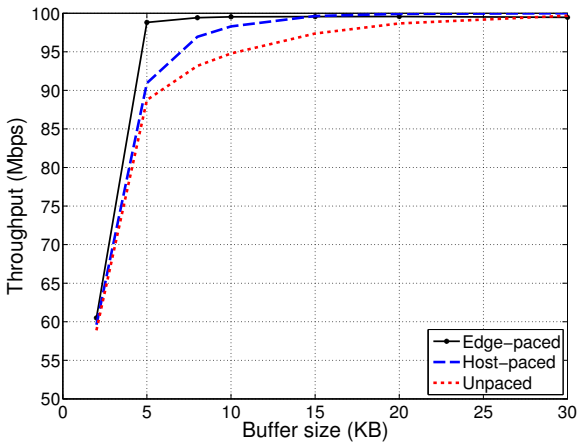


Figure 2: *ns-2* network topology

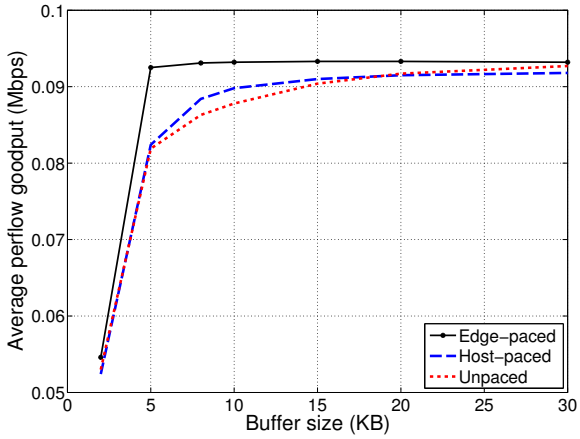
gion bounded above by the arrival curve $A(t)$, and below by the deadline curve $D(t)$. Amongst all feasible exit curves, we have shown that the one which corresponds to the smoothest output traffic is the shortest path between the origin $(0, 0)$ and $(T, D(T))$, as shown in Fig. 1. In our earlier work 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. It is interesting to note that our algorithm for optimal pacing satisfies the properties that it does not space packets out when the link is heavily loaded (since that would cause packets at the tail of the queue to violate their delay bound), and it does space packets out maximally (subject to the delay constraint) when the link load is light. 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 [6].

3. 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 [21] and runs in *ns-2* version 2.28.



(a) Aggregate TCP throughput



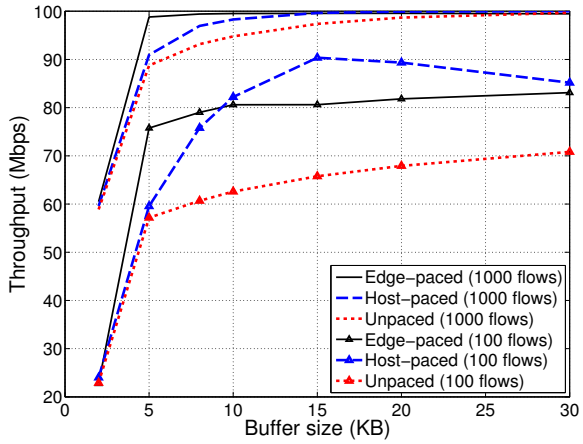
(b) Average per-flow goodput

Figure 3: TCP performance with small buffer link as the bottleneck

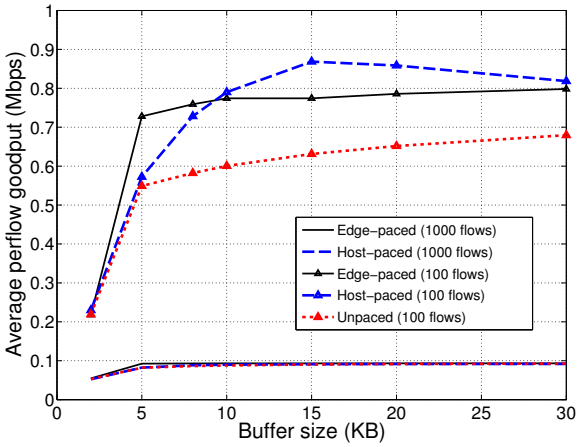
We conducted extensive simulations in *ns-2* for evaluating the effectiveness of traffic pacing. Our simulations were limited to link speeds of 100 Mbps and 1000 flows, which already stretch *ns-2* to the limits of time and memory resources available to us. In what follows we compare the performance of edge pacing with 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 [22], 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.

3.1. Small buffer link as the bottleneck

Our simulations were conducted on the single link dumbbell topology shown in Fig. 2. Ten ingress edge



(a) Aggregate TCP throughput



(b) Average per-flow goodput

Figure 4: TCP performance with 100 and 1000 flows

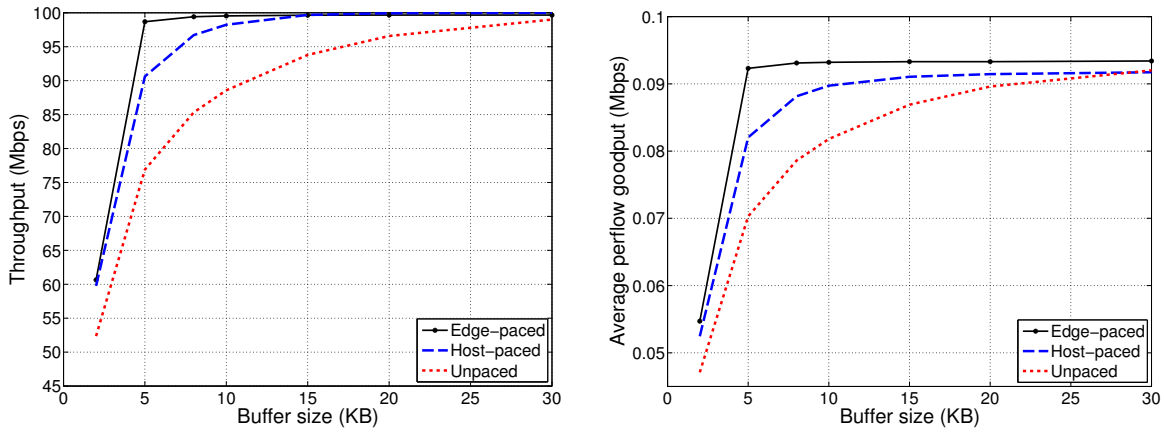
links (E_{S1}-E_{S10}) feed traffic into the core link C_0-C_1, with each edge link in turn fed by hundred access links. Each end-host has one TCP (Reno) agent, and the network therefore simulates 1000 long-lived TCP flows (short-lived flows and different TCP versions are discussed below). Similarly 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.

3.2. Number of TCP 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,



(a) Aggregate TCP throughput

(b) Average per-flow goodput

Figure 5: TCP performance with high-speed access links

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.

3.3. 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.

3.4. Short-lived TCP flows

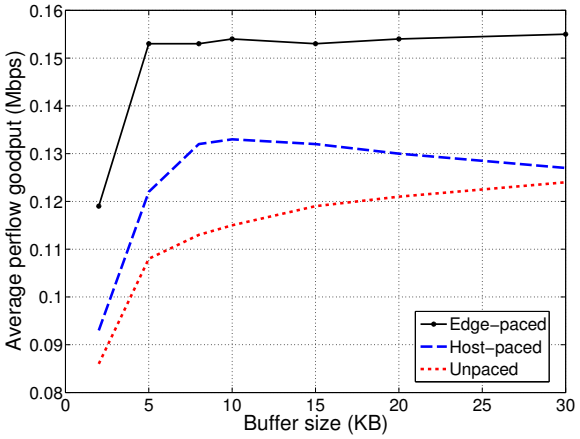
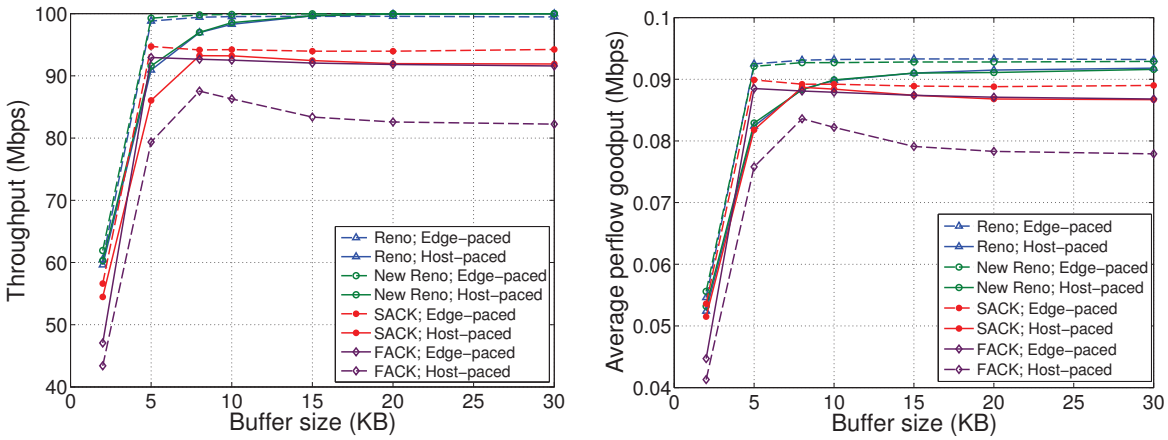


Figure 6: Average per-flow goodput with short-lived TCP flows

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



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

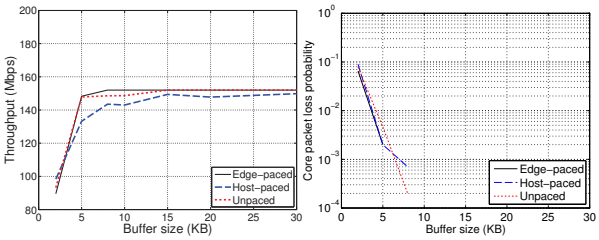
Figure 7: Performance with different TCP variants

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 [23], 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).

3.5. *Different versions of TCP*

We compared the performance of edge and host pacing with three additional variants of TCP (New Reno, SACK and FACK). Simulations in Section 3.1 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.

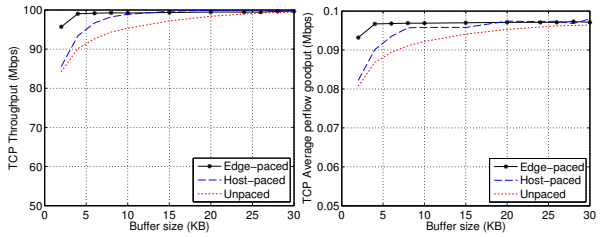
3.6. *Small buffer link not the bottleneck*



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

Figure 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



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

Figure 9: TCP performance with RED queue policy

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.

3.7. Random Early Detection drop policy

We now evaluate the efficacy of our pacing scheme when the core link employs the Random Early Detection (RED) queue drop policy. The simulation settings are the same as section 3.1. We set the RED parameters according to [24]: the maximum and minimum RED drop thresholds are set to the buffer size and 40% of the buffer size respectively. We use the ns2 default values of 0.1 for the maximum drop probability and 0.002 for the queue weight.

Fig. 9 shows that edge pacing performs as well or better than host pacing in terms of both aggregate throughput and average per-flow goodput. We observe from Fig. 9(a) that edge pacing requires only 4 KB of buffering to realize 99 Mbps of aggregate throughput (i.e. to achieve 99% of link capacity), which is less than half of the buffering required when TCP flows are paced at the end-hosts (10 KB). Fig. 9(b) shows that at 6 KB of buffering, unpaced flows obtain 89 Kbps of googput on average, host pacing improves this by 5% (to 94 Kbps), while edge pacing achieves 97 Kbps (9% improvement compared to no pacing). These results show that edge pacing is also effective when the RED drop policy is employed at the core bottleneck link.

4. 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 *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:

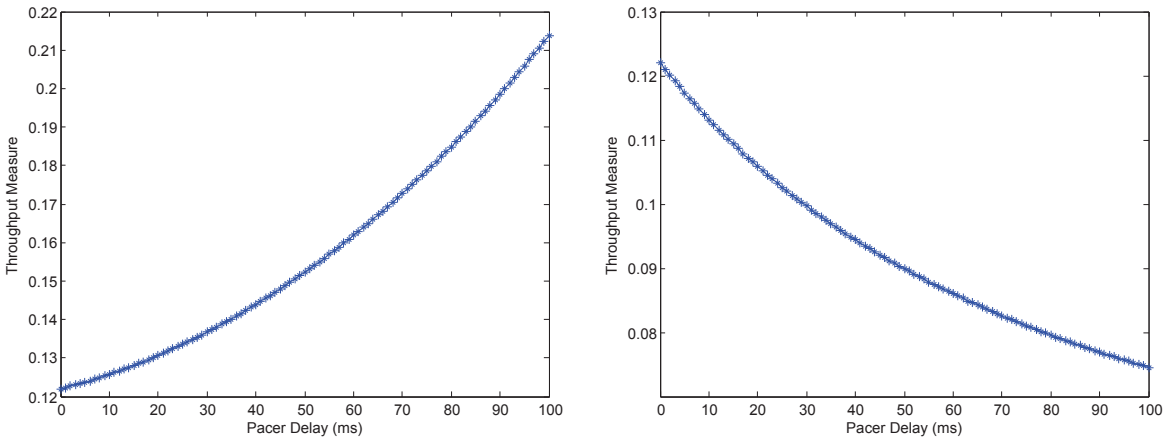
$$T \propto \frac{1}{RTT\sqrt{L}} \tag{1}$$

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.

The pacing delay incurred by a packet can vary between $\epsilon > 0$ (when the link is continuously heavily loaded) and d (when the link is lightly loaded). Quantifying the dependence of pacing delay on (first and second moments of) the link load is non-trivial. We thus make an approximation that the average delay incurred by a random packet is $d/2$ (we acknowledge that this is a simplification to make the analysis tractable), increasing mean RTT to $RTT_0 + d$, where RTT_0 is the round-trip-time without pacing.

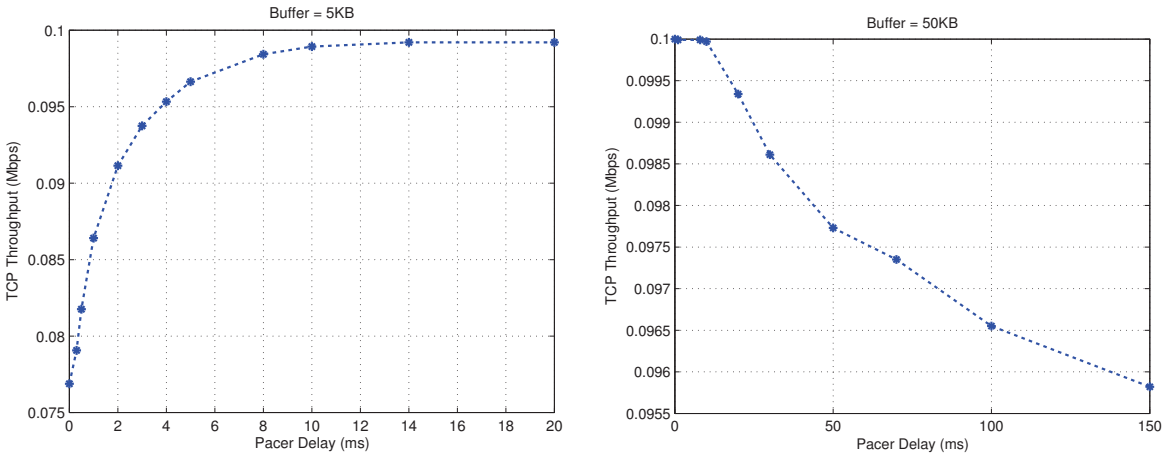
Quantifying loss at a buffer fed by several (paced) TCP sources is however non-trivial. A worst-case assumption that all TCP sources synchronise their bursts (to yield a giant saw-tooth) is unrealistic (especially when thousands of TCP sources share the link) and excessively conservative. We instead resort to the observation (made in our earlier work [25] and by others e.g., [26, 27]) 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 [25] 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



(a) Low load/Small buffer (b) High load/Large buffer

Figure 10: Throughput measure as a function of pacer delay from our analytical model for (a) Low load (small buffer), and (b) High load (large buffer)



(a) Low load/Small buffer (b) High load/Large buffer

Figure 11: Throughput measure as a function of pacer delay from simulations for (a) Low load (small buffer), and (b) High load (large buffer)

burstiness (ratio of standard deviation to mean rate) given by [6]:

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

Further, the loss rate, derived using a bufferless fluid approximation, is obtained from the Chernoff bound as [6]:

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

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. 10 for two cases: Fig. 10(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. 10(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 and bottleneck link buffers. When many TCP flows share bottleneck link buffers, we have shown in [7] (and several other researchers have corroborated [28]) 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. 11(a) the per-flow TCP throughput. It shows that TCP throughput increases with pacing delay, as predicted by our analysis in Fig. 10(a). We now set the core link buffers to 50 KB, and plot in Fig. 11(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. 10(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, though 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. 10(a) is convex while the simulation curve in Fig. 11(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.

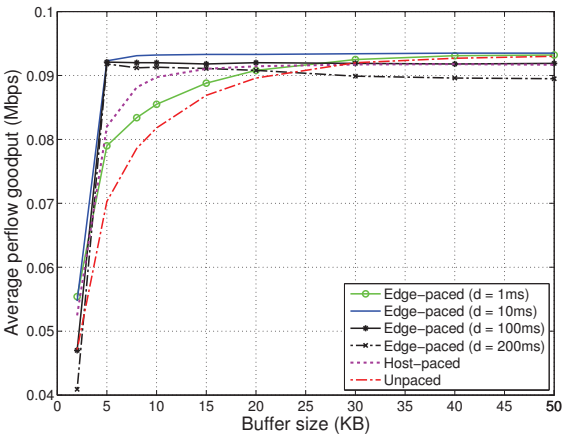
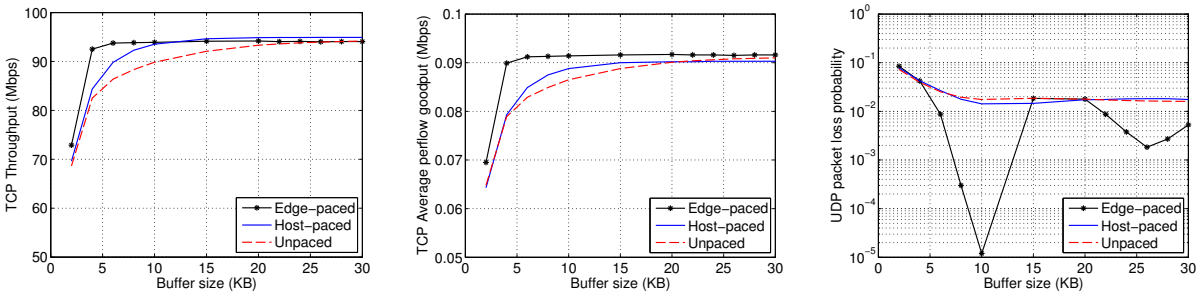


Figure 12: Per-flow TCP goodput for various pacing delay values

In Fig. 12 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.

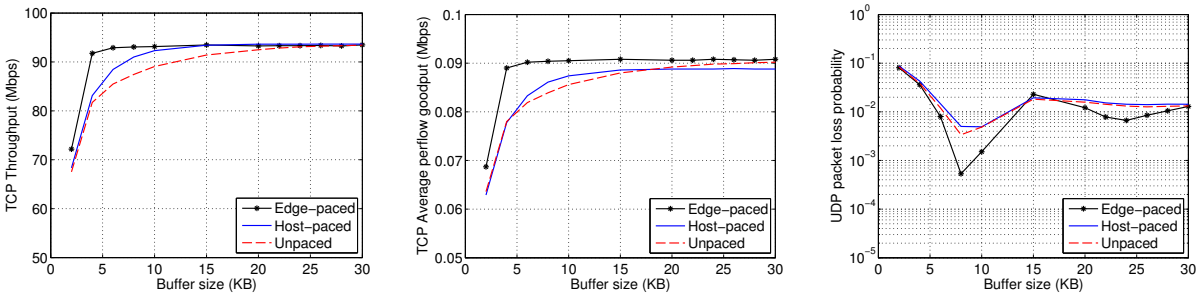
5. Mixed TCP and UDP Traffic

Having demonstrated the benefits that edge pacing offers to TCP flows, in this section we will evaluate via simulations the impact of edge pacing on the joint performance of TCP and UDP traffic when they multiplex at a core link. Observations of traffic in the Internet suggest that TCP constitutes around 90% of the total traffic, with real-time (open-loop UDP) traffic accounting for about 5-10%. We will show that the aggregate throughput and average per-flow goodput for TCP, as well as the packet loss rate for UDP, improve substantially when they are paced at the network edge.



(a) Aggregate TCP throughput (b) Average per-flow goodput (c) UDP loss rate

Figure 13: TCP and UDP performance with poisson UDP



(a) Aggregate TCP throughput (b) Average per-flow goodput (c) UDP loss rate

Figure 14: TCP and UDP performance with fBm UDP

The simulations were performed using the dumbbell topology shown in Fig. 2, as described in previous sections. In all we had 1000 TCP-Reno flows, which is near the limit of what can be simulated in reasonable time within the ns2 simulator. The real-time traffic is modeled as a single UDP flow, which suffices since open-loop traffic can, without loss of generality, be aggregated. There has been much debate in the literature about whether aggregated traffic has short- or long-range dependent characteristics, and so in this paper we will consider both types of models for our aggregated UDP flow. The UDP access and edge links operate at 100 Mbps, and the packet sizes were uniformly distributed in the range $[100, 300]$ Bytes. Other parameters in our simulations were as in Section 3.1. The edge pacer, which is agnostic to the type of traffic, paces both TCP and UDP flows. However, host pacing, when considered, is applied only to TCP flows.

5.1. Poisson UDP

We first consider the commonly used Poisson model for aggregated UDP traffic. The average UDP traffic rate is chosen as 5.5 Mbps, accounting for 5.5% of the bottleneck link capacity, which is consistent with observations in the Internet. This traffic multiplexes with the TCP flows at the bottleneck link, and in Fig. 13 we show the aggregate TCP throughput, the average per-flow TCP goodput, and the UDP loss rate (on log-scale) as a function of the bottleneck link buffer size. The aggregate throughput plots in Fig. 13(a) show that edge-pacing outperforms unpaced and host-paced flows, reaching a saturation throughput of roughly 95 Mbps with only 6 KB of buffering; by contrast, host pacing requires more buffers at the bottleneck (10 KB) to reach saturation throughput, while absence of pacing requires substantially more buffers (more than 20 KB). A similar pattern in per-flow throughput is corroborated in Fig. 13(b) that shows that at 6 KB of buffering, edge-pacing gives 91 Kbps to each TCP flow, whereas host-pacing gives around 85 Kbps and no pacing only around 82 Kbps. These plots corroborate that edge pacing outperforms host pacing (and no pacing) for TCP flows even in the presence of real-time UDP traffic in the network.

We now explore the impact of pacing on UDP traffic – Fig. 13(c) shows the UDP loss rate (on log scale) as the buffer size varies. Our first observation is that the loss rate is non-monotonic in buffer size. This phe-

nomenon is not surprising and arises due to the interaction between closed-loop TCP traffic and open-loop UDP traffic at the bottleneck link with small buffers – we refer the reader to our detailed study in [7] that provides qualitative and quantitative explanations of this phenomenon. The observation from the figure that is more pertinent to our study in this paper is that the UDP loss is same or lower with edge pacing than with host pacing or no pacing – indeed, in certain regimes (e.g. 5-15 KB and 20-30 KB buffers), UDP loss is lower by multiple orders of magnitude, corroborating that edge pacing is also beneficial for UDP traffic.

5.2. fBm UDP

We now evaluate the benefits of edge pacing when UDP traffic has long-range dependent properties. For our simulations we generated fractional Brownian motion (fBm) traffic at an average rate of 6 Mbps (6% of the bottleneck link capacity) with a Hurst parameter at $H = 0.85$ representing a relatively high degree of self-similarity. The fluid traffic was generated using the novel filtering method of [29] to obtain long sequences from an underlying Norros' self-similar traffic model [30], packetized as per our method in [6]. In Fig. 14 we show the aggregate TCP throughput, average per-flow TCP goodput, and the UDP packet loss as a function of buffer size.

The aggregate and per-flow TCP throughputs in Fig. 14(a) and Fig. 14(b) show that edge-pacing again outperforms host-pacing and no-pacing, achieving peak throughput for as low as 6 KB buffers, much as in the case of Poisson UDP traffic. The UDP loss rate is shown in Fig. 14(c), and once again shows similar qualitative behavior to what was observed for Poisson traffic before, namely that (a) loss is non-monotonic in buffer size, irrespective of whether traffic is paced or not (explained by our previous work [7]), and (b) edge-pacing reduces loss for UDP traffic, by as much as an order of magnitude in some regimes. These plots corroborate that edge pacing is beneficial for both TCP and UDP traffic, irrespective of whether the UDP traffic has long-range dependent characteristics or not.

5.3. Impact of Pacer Delay

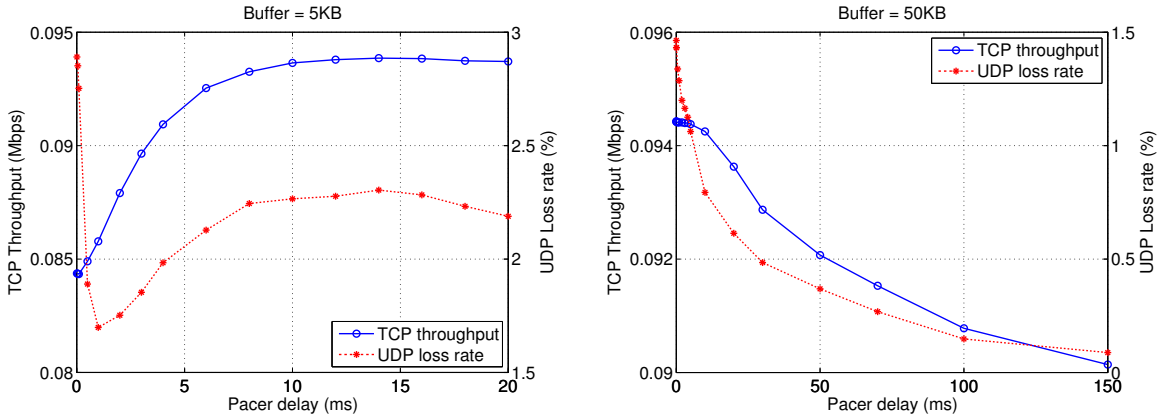
We now study how the choice of edge pacer delay impacts on TCP and UDP performance. Recall that in Section 4 we had shown that in (small buffer) regimes, TCP benefits from larger pacing delay, whereas in high load (large buffer) regimes larger pacing delay can impact TCP performance negatively. In this section we corroborate this behavior in the presence of UDP traffic, and also investigate how UDP loss is affected by pacing delay in both regimes.

In Fig. 15 we plot the per-flow TCP throughput (left axis) and UDP loss (right axis) as a function of pacer delay. In Fig. 15(a) we set buffer size very small (5 KB), and observe that TCP throughput rises monotonically with pacing delay, much as we had observed in Fig. 11 earlier. As argued before in Section 4, this is because at low loads (induced by TCP's reaction to high network loss arising from very small buffers), pacing helps TCP reduce its burstiness and hence loss, allowing its window (and throughput) to grow more. Simultaneously, the right axis shows that UDP loss first falls with pacing delay, and then grows. This can be explained as follows: for small pacing delay (sub-millisecond), UDP traffic benefits from the smoothing and experiences reduced loss; however, as pacing delays grow larger, TCP ramps up its throughput aggressively, leaving less capacity to serve UDP traffic at the bottleneck link, thereby causing its loss to increase. It is seen that beyond about 14 KB, TCP throughput has saturated, and UDP therefore experiences slightly reduced loss as pacing delay increases. We acknowledge that our explanation is only qualitative; however, developing a quantitative model to capture the complex interactions between UDP and TCP dynamics in the presence of pacing turned out to be intractable and beyond the scope of this paper.

When load is higher (due to larger buffers of 50 KB), we observe from Fig. 15(b) that TCP throughput decreases with pacing delay; as argued in Section 4, this is because the increase in RTT from pacing over-rides the benefits of burstiness reduction from pacing. Correspondingly, we can see on the right axis that UDP loss falls monotonically, due to both increase in available bandwidth (that was given up by TCP) and reduction in burstiness from pacing. These plots tell us that in a mixed TCP-UDP scenario, TCP pacing delay should be tuned to load conditions, while UDP benefits from increased pacing delay (at the cost of increasing end-to-end delays for the application).

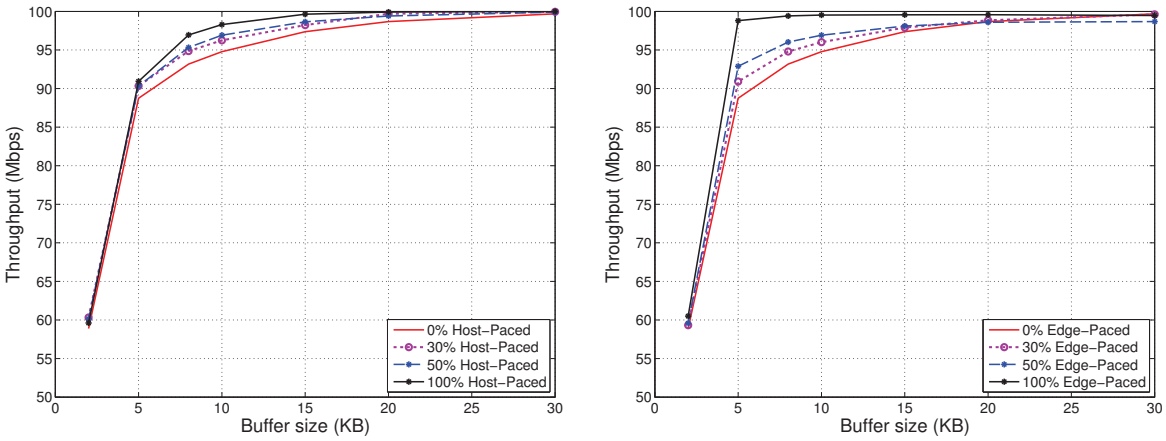
6. Practical Deployment of Pacing

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



(a) Low load/Small buffer (b) High load/Large buffer

Figure 15: Throughput and Loss measure as a function of pacer delay from simulations (Poisson UDP rate 5.5 Mbps) for (a) Low load / small buffer, and (b) High load / large buffer

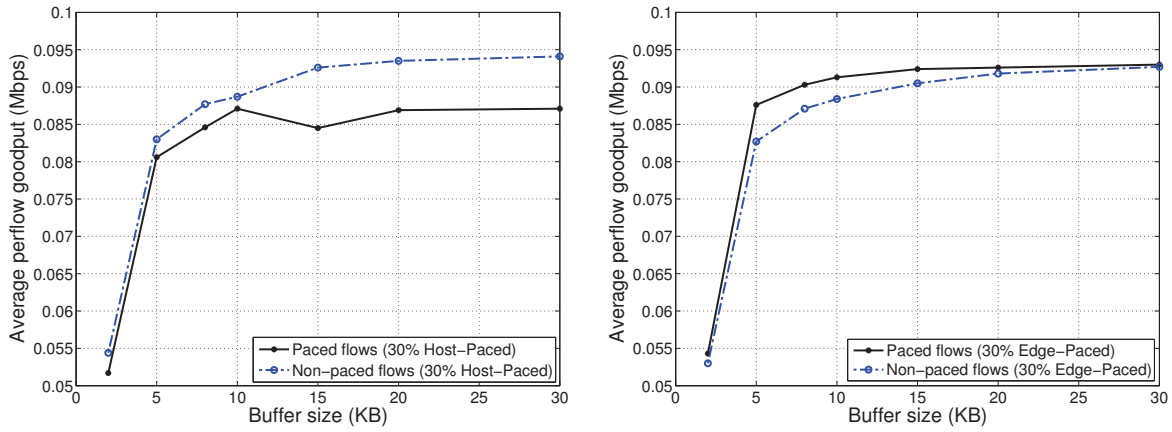


(a) Host pacing (b) Edge pacing

Figure 16: Aggregate bottleneck link throughput versus buffer size for varying fractions of pacing deployment at (a) Hosts and (b) 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. 16 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. 16(a) with edge pacing in Fig. 16(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. 17 we compare the average per-flow goodput of paced versus unpaced flows. Fig. 17(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 [15, 14] 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) Host pacing

(b) Edge pacing

Figure 17: Per-flow goodput versus bottleneck buffer size for paced and unpaced flows with fractional deployment of pacing at (a) Hosts and (b) Edge nodes

a substantial disincentive for users to adopt host pacing. By contrast, Fig. 17(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 [6] 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.

7. Conclusions

Energy density concerns in modern high-speed routers are driving the trend towards photonic integrated switching platforms [31] with reduced buffering capability. In networks with such small buffers, end-to-end performance of TCP/UDP 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/UDP performance in emerging all-optical or hybrid-optical core networks with small buffers.

8. References

[1] H. H. Gharakheili, A. Vishwanath, V. Sivaraman, Edge versus Host Pacing of TCP Traffic in Small Buffer Networks, in: Proc. IFIP Networking, USA, May 2013.

[2] G. Appenzeller, I. Keslassy, N. McKeown, Sizing Router Buffers, in: Proc. ACM SIGCOMM 2004, USA, 2004.

[3] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, T. Roughgarden, Routers with Very Small Buffers, in: Proc. IEEE INFOCOM, Spain, 2006.

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

[5] A. Vishwanath, V. Sivaraman, M. Thottan, Perspectives on Router Buffer Sizing: Recent Results and Open Problems, ACM SIGCOMM Computer Communication Review 39 (2) (2009) 34–39.

[6] V. Sivaraman, H. Elgindy, D. Moreland, D. Ostry, Packet Pacing in Small Buffer Optical Packet Switched Networks, IEEE/ACM Transactions on Networking 17 (4) (2009) 1066–1079.

[7] A. Vishwanath, V. Sivaraman, G. N. Rouskas, Anomalous Loss Performance for Mixed Real-Time and TCP Traffic in Routers with Very Small Buffers, IEEE/ACM Transactions on Networking 19 (4) (2011) 933–946.

[8] E. M. Wong et al., Towards a Bufferless Optical Internet, IEEE/OSA Journal of Lightwave Technology 27 (4) (2009) 2817–2833.

[9] F. Farahmand, Q. Zhang, J. P. Jue, A Feedback-based Contention Avoidance Mechanism for Optical Burst Switching Networks, in: Proc. Workshop Opt. Burst Switching, USA, 2004.

[10] H. Boyraz, N. Akar, Rate-controlled optical burst switching for both congestion avoidance and service differentiation, Opt. Switch. Netw. 2 (4) (2005) 217–229.

[11] Y. Cai, B. Jiang, T. Wolf, W. Gong, A Practical On-line Pacing Scheme at Edges of Small Buffer Networks, in: Proc. IEEE INFOCOM, USA, 2010.

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

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

[14] D. X. Wei, P. Cao, S. H. Low, TCP Pacing Revisited (2006). URL `http://www.cs.caltech.edu/weixl/research/\\summary/infocom2006.pdf`

[15] A. Aggarwal, S. Savage, T. E. Anderson, Understanding the Performance of TCP Pacing, in: Proc. IEEE INFOCOM, Israel, 2000.

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

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

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

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

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

[21] D. X. Wei, A TCP Pacing Implementation for NS2 (2006). URL `http://netlab.caltech.edu/projects/\\ns2tcplinux/ns2pacing/index.html`

[22] N. Dukkipati, N. McKeown, Why Flow-Completion Time is the Right Metric for Congestion Control, ACM SIGCOMM Computer Communication Review 36 (1) (2006) 59–62.

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

[24] General main RED parameters. URL `http://www.cs.technion.ac.il/Courses/Computer-Networks-Lab/projects/spring2003/ns1/Net\%20Site/Source/RedParam.htm`

[25] A. Vishwanath, V. Sivaraman, 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.

[26] A. Lakshmikantha, C. Beck, R. Srikant, Impact of File Arrivals and Departures on Buffer Sizing in Core Routers, IEEE/ACM Transactions on Networking 19 (2) (2011) 347–358.

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

[28] L. Andrew, T. Cui, J. Sun, M. Zukerman, K. Ho, S. Chan, Buffer Sizing for Nonhomogeneous TCP Sources, IEEE Communications Letters 9 (6) (2005) 567–569.

[29] D. Ostry, Synthesis of accurate fractional gaussian noise by filtering, IEEE Transactions on Information Theory 52 (4) (2006) 1609–1623.

[30] I. Norros, On the use of Fractional Brownian Motion in the Theory of Connectionless Traffic, IEEE J. Selected Areas in Comm. 13 (6) (1995) 953–962.

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

Hassan Habibi Gharakheili is currently a Ph.D. candidate in the School of Electrical Engineering and Telecommunications at the University of New South Wales in Sydney, Australia. He received his B.Sc. and M.Sc. degrees from Sharif University of Technology, Tehran, Iran in 2001 and 2004 respectively. His research interests include software defined networking, optical networking and QoS.

Arun Vishwanath is a Research Scientist at IBM Research Australia. He received the Ph.D. degree in Electrical Engineering from the University of New South Wales in Sydney, Australia, in 2011. He was a visiting Ph.D. scholar in the Department of Computer Science at North Carolina State University, USA in 2008. His research interests include software defined networking and energy-efficient networking.

Vijay Sivaraman received his B. Tech. from the Indian Institute of Technology in Delhi, India, in 1994, his M.S. from North Carolina State University in 1996, and his Ph.D. from the University of California at Los Angeles in 2000. He has worked at Bell-Labs as a student Fellow, in a silicon valley start-up manufacturing optical switch-routers, and as a Senior Research Engineer at the CSIRO in Australia. He is now an Associate Professor at the University of New South Wales in Sydney, Australia. His research interests include Optical Networking, packet switching and routing, network architectures, and sensor networks for the environment, healthcare and sports monitoring.