TCP Pacing Revisited

David X. Wei EAS, Caltech
Pei Cao CS, Stanford
Steven H. Low EAS, Caltech

***Abstract*— TCP pacing promises to reduce burstiness of TCP traffic and alleviate the impact of under-buffered routers and switches on flow throughput. However, current research literatures have not always agreed on the overall benefits of pacing. In this paper, we re-examine the benefits and drawbacks of TCP pacing in light of new TCP variants, new application requirements, and trends in router technologies. We found that pacing primarily has three effects: reduced burstiness of traffic, increased synchronization amnog the flows and fragmented SACK blocks in a flow. We analysis how these factors play out for different TCP implementations (Reno, NewReno, SACK, FACK) and new high-speed TCP protocols (BIC-TCP and FAST). We conclude that TCP pacing brings significant benefits for many applications, and though paced flows sometimes suffer in performance when competing with non-paced flows, there are enough incentives for applications to migrate from using non-paced TCP to paced TCP.**

I. Introduction

A. Motivation

TCP (Transmission Control Protocol) pacing evens out the transmission of a window of packets over a round-trip time (RTT), so that packets are injected into the network at the desired rate of congestion_window_size/RTT. It was initially suggested by Zhang el al.[26] to reduce burstiness of TCP traffic caused by ACK compression. Motivated by throughput improvement over simulated satellite links, the report [17] proposes pacing over the entire lifetime of a TCP connection. However, a more detailed simulation study in [6] concludes that TCP pacing results in lower throughput and higher latencies in many situations. There seems to be no concensus in the research community on whether TCP should pace. Yet, experiments in real high-speed WAN (wide-area networks) showed that pacing significantly improves the overall throughput of parallel TCP transfers [16].

In this paper, we ask the questions: 1) whether we should pace, and if so, 2) how to migrate from the current state where (we assume) virtually no TCP flow paces to an ideal state where virtually all TCP flows pace. We believe TCP pacing deserves a fresh look because:

- As link speed increases, designing routers with buffer sizes equal to bandwidth-delay product is increasingly difficult, as pointed out in [7]. By minimizing the burstiness in traffic, TCP pacing reduces the impact of under-buffered switches and routers on the throughput of TCP flows.
- TCP Reno is no longer the most prevalent TCP implementation. TCP NewReno, SACK and FACK have all been implemented in common operating systems. TCP SACK is turned on by default in both Windows and Linux. Yet, existing studies comparing TCP pacing and non-pacing focused only on the TCP Reno implementation, leaving unanswered the question of how pacing performs under other implementations.
- New TCP variants have been recently proposed to increase TCP's performance in high-speed WANs, e.g., [10], [25], [14]. These variants use congestion avoidance algorithms that are more aggressive than the classic additive-increase algorithm, potentially introducing more bursty traffic, but also recovering from losses faster. TCP pacing may have very different interactions with these variants than with classic TCP Reno.
- For many important distributed applications, the performance criteria is not the aggregate throughput of TCP flows, but rather the throughput of the slowest flow. Applications that perform scatter/gather type of communications, such as stripped file system (e.g. Google file system [11]) and shared-nothing distributed databases, are bottlenecked on the slowest flow. In high-speed LAN, these flows last a few hundred RTTs, making the applications highly vulnerable to short-term unfairness in TCP, an issue that TCP pacing can address.

B. Summary

Our conclusions are depicted in Figure 1, which describes the performance (to be made precise in the following sections) of paced flows and that of nonpaced flows when they share the same network, as functions of the fraction π of paced flows. [1] Our simulations examine the performance of paced and nonpaced flows at $\pi = 0\%$ (no flow pace), $\pi = 50\%$, and $\pi = 100\%$ (all flows pace), using TCP Reno, SACK, BIC, and FAST. Our preliminary conclusions are as follows.

Should we pace? The answer is a resounding "Yes" from a performance perspective. The performance (of paced flows) when all flows pace is generally much better than the performance (of nonpaced flows) when none pace, i.e., $B > A$ in Figure 1.

Can we migrate? One of the main drawbacks of pacing that is discovered in [6] is that the performance of paced flows is often lower than that of nonpaced flows when they share the same network, i.e., $D < C$ in Figure 1. We make three remarks. First, we view this as an migration issue, not one that determines whether we should pace. If paced flows perform as well or better than nonpaced flows when they share the same network, then the system will naturally migrate itself from $\pi = 0\%$ to $\pi = 100\%$. Otherwise, other mechanisms are needed to facilitate the migration, e.g., by raising the performance curve for paced flows or through policy. Second, while our

[1]The figure is not meant to be quantitatively accurate but captures the key message we have learnt from a comprehensive set of simulation experiments.

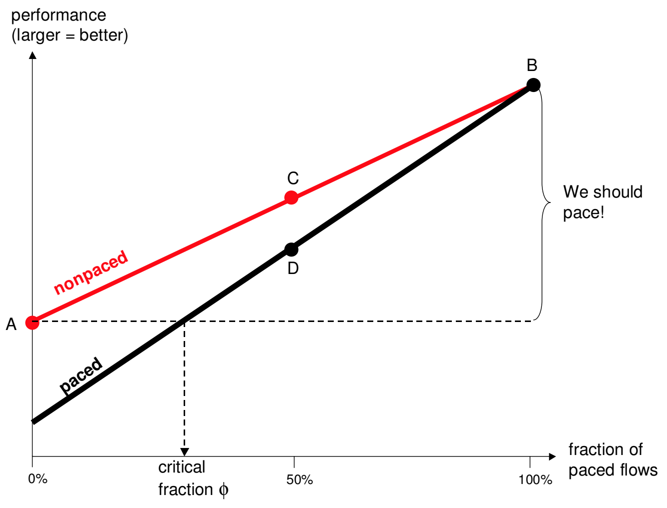

Fig. 1. Qualitative conclusions. $B > A$: we should pace. $D < C$: disincentive to migrate. $D > A$ and $B > C$: incentives to migrate.

results are consistent with those in [6], the performance gap between paced and nonpaced flows is generally smaller in our experiments that use an improved pacing algorithm, and with new TCP variants. Finally, note that there is a critical fraction ϕ above which all flows gain in performance, when some pace, compared with when no flow paces, e.g., $D > A$, even though those that remain unpaced gain more ($C > D$). This provides an incentive to migrate out of $\pi = 0\%$ to a fraction of paced flows that exceeds the critical fraction ϕ. Moreover, since $B > C$, even nonpaced flows will gain in performance when all of them pace. This provides an incentive for all to migrate to the point $\pi = 100\%$.

C. Related work

Our initial interest in TCP pacing came from the observation that high-speed TCP protocols sometimes have very poor throughput due to burstiness coupled with under-buffered routers and switches on the network path [24]. Since pacing is a natural way to control burstiness of the flows, we became interested in how pacing interacts with high-speed TCP protocols.

The two papers most relevant to our study are [17] and [6]. In [17], the authors examined the benefits of TCP pacing for the satellite environment, which, similar to the high-speed WAN networks, have large bandwidth-delay products that exceeds routers' buffer sizes. The paper evaluated the combination of pacing with Classic TCP, TCP Reno and TCP FACK, and concluded that pacing can improve the throughput of single flow and the aggregate throughput of multiple flows by 20%. However, the paper did not study what happens when the paced flows compete with non-paced flows, and the paper focused on low-buffer environments only.

The study in [6] took a more comprehensive look at pacing in TCP Reno, comparing throughput and average latencies of paced and non-paced flows under different bandwidth-delay products and router buffer configurations. The study particularly looked into the relative performance of pacing and non-pacing when paced flows and non-paced flows compete for a bottleneck link. The study reached the conclusions that pacing results in lower throughputs and higher latencies in most of scenarios that the authors examined.

The two papers inspired our work, as they seem to reach opposite conclusions about TCP pacing. We replicated many of the scenarios in [6] and obtained the same results as reported in [6]. However, unlike [6], which only investigated TCP Reno, we also looked into pacing with other AIMD TCP implementations including NewReno, SACK and FACK. We found that the impacts of pacing are more multi-faceted, that paced flows do not inherently lose to non-paced flows. A simple change in pacing's behavior during slow-start eliminates its lower performance compared to TCP non-pacing. If the SACK block fragmentation issue does not come into play, as in the case of TCP Reno and TCP NewReno, pacing achieve similar performance to non-pacing in competition cases.

Our focus on "worst-flow latency" is shared by many in the high-performance computing community. It's well known that, to cope with the short-term unfairness of TCP, parallel FTPs (i.e. FTP with parallel TCP connections) should actively move data from slow flows to fast flows. However, this is only possible when the flows originate from the same host. In the Data Reservior project [21], where a collection of 26 hosts in Japan are used to perform parallel iSCSI transfers to a collection of 26 hosts in Baltimore, MD over an OC-12 link across the Pacific Ocean, such data movement is not possible. As a result, the authors in [15] report that the latency of the slowest flow is three times that of the fastest flow. The surprise in [15] is that the total transfer latency, which is bounded by the latency of the slowest flow, is significantly improved when the hosts use the FastEthernet NIC (100Mb/s) instead of the GigaBitNIC (1Gb/s). In effect, the FastEthernet NIC in that case paced the flows, albeit via an inflexibile hardware mechanism.

The type of pacing studied in this paper is applied throughout the lifetime of the flow. Pacing as a technique was initially suggested by Zhang et al. [26] to reduce burstiness of TCP traffic caused by ACK compression. Over the years, researchers have also proposed using pacing in cases when TCP might generate a burst of packets, for example, after a packet loss in TCP NewReno [12] and TCP Forward-Acknowledgement [19], or when a persistent HTTP connection restarts after idle time [22]. However, these studies investigated pacing when enacted seletively, instead of being applied at all times over the lifetime of a connetion.

A number of commercial products explicitly tort pacing as a mechanism to improve interactive-flow latencies and enforce rate control. Devices from Packeteer Inc. implement TCP pacing by pacing the delivery of acknowledgements to senders[13]. Other commercial products enforces pacing by explicitly buffering packet bursts.

II. Performance of TCP Pacing

In this section, we study the performance of TCP pacing when all the flows in the network are paced TCP flows.

A. Performance metrics

We exam the performance of pacing with two metrics: maximum latency and aggregate throughput.

Aggregate throughput has been heavily used in examination of TCP performance in previous research. Maximum latency is a new metric we introduce to analyze the performance of TCP. It reflects the need of the distributed applications which use multiple TCP flows in communication and are bottlenecked by the throughput of the slowest flow.

1) Worst-flow latency: While the traditional metrics for TCP protocol performance has been aggregate throughput and average flow latency such as HTTP response time, we argue that a third metric, "worst-flow latency", should also be considered. "Worst-flow latency" is a measure of fairness among a set of flows that start around the same time and have the same RTT. Simply put, a "worst-flow" latency is the latency of the slowest flow to finish the transfer, where N flows of the same RTT start at the same time and each have to finish transfer of B bytes.

"Worst-flow latency" is important for an increasingly common type of distributed application platforms: clustered systems. Over the last decade, improvements in LAN and PC technologies have resulted in a proliferation of cluster computing, that is, connecting a cluster of PCs via high-speed LAN for high-performance computing. Since an individual PC often has limited computation and I/O power, a common way of achieving high-throughput from the cluster is to "stripe" data across the machines and perform computation on a data-driven basis across multiple machines. The results are then "gathered" back to a master node. Examples of such systems are clustered file systems such as the Google File System [11] and the Stanford Linear Acceleration Centerl's xrootd system [5], "shared-nothing" parallel database architectures such as [4] and high-performance remote I/O architectures such as the parallel iSCSI [21].

These parellel systems all use TCP/IP as the communication protocol. The traditional wisdom in the systems' community has been that TCP is fair, and therefore data and communications should be spread as evenly among the nodes as possible. For example, parallel file systems break file data into 64KB chunks and put each chunk on a different node. Reading 2MB of data from a file are accomplished by sending read requests to 32 nodes, each sending 64KB of data to the requesting node. From the requesting node's perspective, the latency of the 2MB file read is the latency it takes the *slowest* flow to finish the 64KB transfer, not the average flow.

Thus, short-term fairness or the lack thereof in TCP has a significant impact on these parallel applications, and should be considered as a metric of the protocol as much as aggregate throughput.

2) Aggregate throughput: Aggregate throughput is defined as the sum of the individual throughputs among a set of parallel flows. It reflects how TCP utilizes the bottleneck capacity efficiently. This metric has been heavily used by the research community. However, the aggregate throughput alone cannot reflect the fairness among TCP flows. It cannot predict the maximum latency of an TCP based application, either.

Besides maximum latency and aggregate throughput, there are many other important metrics for TCP performance. For example, queueing delay variation and loss rate are very important for many applications. But the current literature have reached consistent conclusions that pacing help to improve the performance in terms of these metrics. We hence focus on maximum latency and aggregate throughput. [6] uses average latency among all the flows as a performance index. This is similar to aggregate throughput of all flows.

B. Performance: worst-flow latency

In this subsection, we first show, by calculation, that TCP may take thousands of round trips to converge to fairness and hence the worst-flow throughput can be very low for a long time. Pacing greatly improves the startup fairness and hence the worst-flow throughput.

1) Convergence time of fairness: [9] proves that AIMD congestion control algorithms converge to fairness with a synchronization model. However, the time for the convergence depends on several factors and can be very long.

Take Reno as an example. If Reno flows are non-paced, some of them experience loss events earlier in the slow start phase, and the other experience loss events later. For simplicity, we only study the case where there are two sets of flows. One set of the flows (denoted by F_1) experiences loss events and exit slow start one RTT earlier than the other set of the flows (denoted by F_2). We assume that all flows have no timeout and slow start only happens in the startup period. Currently, we also assume that the loss events are synchronized in the congestion avoidance phase. That is, all the flows in congestion avoidance see a packet loss event if one flow sees a packet loss event.

Let N be the number of homogeneous Reno flows sharing a bottleneck with capacity cN and buffer size BN. The round trip propagation delay is d. Let the window size of the i-th flow, at the end of the k-th congestion epoch[2] to be $w_k^{(i)}$ (before halving). Let the window size at the end of slow start to be $w_0^{(i)}$. Because there is a packet loss event at the end of each congestion epoch, which indicates that the buffer in the router is full, we have:

$$\forall k \geq 0 : \sum_{i=1}^{N} w_k^{(i)} = (cd + B)\, N \tag{1}$$

Let $r \in$ be the ratio of flows that exits slow start earlier, we have $|F_1| = rN$ and $|F_2| = (1-r)\, N$.

Since flows in slow start double their congestion windows every RTT, we have

$$\forall i \in F_1, \forall j \in F_2 : w_0^{(i)} = \frac{1}{2} w_0^{(j)} \tag{2}$$

Combining (2) and (1), we have $w_0^{(i)} = \frac{cd+B}{(2-r)}$ and $w_0^{(j)} = \frac{cd+B}{\left(1-\frac{r}{2}\right)}$.

[2]*Congestion epoch* is defined as the time between two consecutive loss events. See [8] for details.

Assuming all the loss events are synchronized in congestion avoidance phase, we have $w_k^{(i)} = \frac{w_{k-1}^{(i)}}{2} + \frac{cd+B}{2}$ for $\forall k \geq 1$. This recursive equation can be solved into

$$w_k^{(i)} = cd + B + \frac{1}{2^k}\left(w_0^{(i)} - cd - B\right) \quad (3)$$

Now, we calculate the time it takes for the i-th flow to converge to 95% of its fair share. Let K denote the number of congestion epochs for the length of convergence. We have:

$$w_K^{(i)} = \frac{19}{20}(cd + B) \quad (4)$$

Solving the equations (3) and (4), we have:

$$K = \left\lceil \log_2\left(20 - \frac{20}{(2-r)}\right)\right\rceil \in [0, 4] \quad (5)$$

(5) shows the number of congestion epochs that Reno takes to converge to fairness. This is equivalent to $\frac{cd+B}{2}K$ RTTs in time. K is independent of capacity, delay, buffer size and number of flows and the convergence time increases linearly as the bandwidth delay product increases.

When the bandwidth delay product is large, the convergence time can be very long. For example, when c=100Mbps and d=120ms, the convergence time is more than 2000 RTT if one flow exits slow start one RTT earlier than the others.

Note that this calculation is based on the assumption that the loss events are synchronized in congestion avoidance phase. [20] points out that the loss event rates observed by two flows may be different. The slower flows may observe higher loss event rates than the faster flows do. This phenomenon only makes the convergence time even longer.

The new variants of loss-based TCP congestion control algorithms introduce more frequent congestion loss than Reno. Their loss epoch is hence much shorter than Reno. This is good for convergence. However, most of these new protocols take much more number of loss epochs to converge to fairness. The unfairness introduced by slow start still have a long-lasting effect, as pointed out by [18].[3]

With TCP pacing, the loss events in slow start are very likely synchronized within one RTT and the flows with the same RTT have similar initial congestion window size when they enter congestion avoidance. This greatly helps the fairness in the first several congestion epochs. If we take the throughput of the slowest flow as our performance index, paced TCP is better than non-paced TCP, although the aggregate throughput of paced TCP may be smaller than non-paced TCP, due to synchronization effects.

[6] has observed that pacing can synchronize packet loss events among flows in slow start. In the rest of this section, we give three examples to show that pacing leads to better fairness and smaller latency for the slowest flows. To eliminate pacing's advantage in sub-RTT level control, we set the bottleneck buffer size to be equal or more than bandwidth delay product in the first two simulations. And we examine how the results vary with buffer size in the third simulation.

[3]For delay-based protocols like Vegas and FAST, they do not depend on loss event to converge to fair share. However, pacing still improves their fairness slightly due to pacing's effect of reducing queue variation.

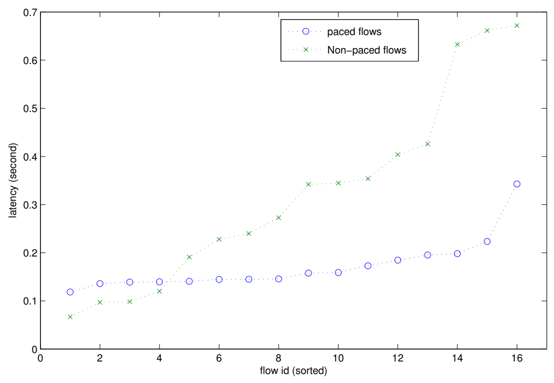

Fig. 2. [Scenario 1] The latency of 16 parallel SACK flows in LAN. Each flow sends 1MB data.

2) Scenario 1: SACK flows in LAN: In this scenario, we have 16 Reno (with SACK) flows sharing 1Gbps bottleneck link with 3ms buffer. The RTT is 2.5ms. The access links are all 1Gbps with infinite buffer. The flows' starting time is uniformly distributed in $(0, 0.5\text{ms})$. We also introduce exponential on-off traffic with a rate of 1% of the capacity. We ran the simulations three times and take the Such scenario is very typical in distributed computation, where a set of homogeneous flows are running in a LAN to exchange the computation results in the distributed system. The flows start almost at the same time and there is a small amount of noise as cross traffic.

First, we measure the latency for each flow to transmit its first 1 megabytes of data (725 standard packets). Figure 2 is the latency for each flow.[4] From this figure, we can see that although almost half of the non-paced TCP flows have smaller latency than the fastest paced TCP flows, the slowest non-paced flow has a latency equal to two times of the paced flow's latency. So, in terms of the slowest flow's throughput, paced TCP has much better performance.

We run the same simulation for 10 seconds. The throughput of the fastest flow and the throughput of the slowest flow are shown in Figure 3. We can see that such advantage of paced flows lasts for the first 4 seconds of the simulation. This is equivalent to 2800 RTTs. Note that all flows in this simulation exit slow start in the first 0.1 seconds. But the unfairness introduced by slow start lasts much longer and has significant impact on the latency of the slowest flow.

3) Scenario 2: BIC-TCP and FAST in WAN: In this scenario, we simulate TCP flows running over long distance network. The scenario has 16 flows sharing a bottleneck with capacity of 1Gbps and a buffer equal to bandwidth delay product. The round trip propagation delay is 120ms. The flows' starting time is uniformly distributed in $(0, 120\text{ms})$. We also introduce exponential on-off traffic with a rate of 10% of the capacity.

This scenario is similar to the high energy physics network ([2]) which is running between Chicago and Geneva.

[4]We run the simulation three times with different random seeds. The result shown in the figures are the average over multiple runs.

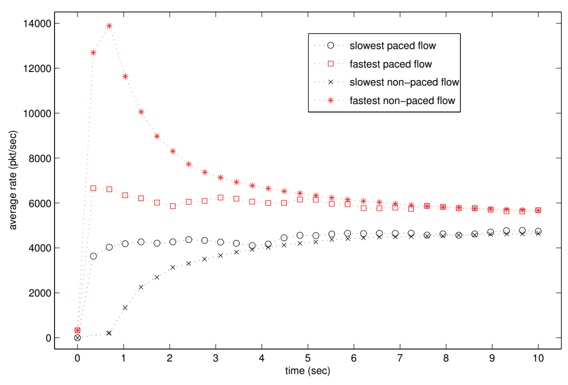

Fig. 3. [Scenario 1] The throughput of the fastest flow and the slowest flow among 16 parallel SACK flows in LAN.

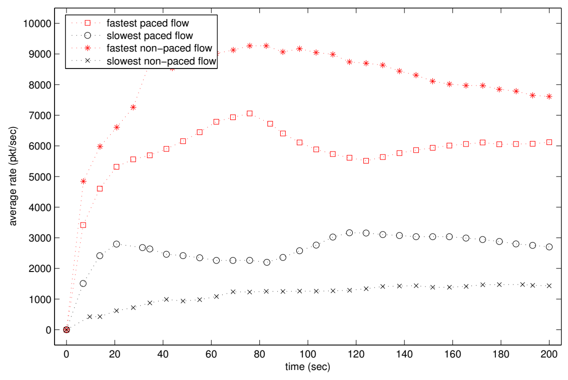

Fig. 4. [Scenario 2] The throughput of the slowest flow and the fastest flow among the 16 BIC-TCP flows in WAN.

We run the simulation with a loss-based high speed TCP variant (BIC-TCP [25]) and a delay-based high speed variant (FAST [14]). The NS codes for BIC-TCP and FAST are from [1] and [3] correspondingly. All the protocol parameters are set to default values.

Figure 4 shows the throughput of the slowest BIC-TCP flow and fastest BIC-TCP flow over time. The slowest BIC-TCP flow without pacing is about half of the throughput of the slowest paced flow. This means that non-paced BIC's maximum latency is about two times of the paced BIC's maximum latency.

The simulation runs for 200 seconds and the gap still exists. This agrees with the observation in [18]. Although the BIC-TCP has much shorter congestion epochs than Reno, it still takes a long time to converge to the fairness point if the flows are very unfair when they exist slow start.

Figure 5 shows the throughput of the fastest flow and the slowest flow of FAST. As an equation-based protocol with delay signal, FAST converges much faster to fairness. However, we observe a constant small gap between the fastest flow and the slowest flow in non-paced case. The reason behind this gap is that some of the flows send a window of packets in several large bursts and some of the flows send the packets in smaller bursts. Those who send packets in large

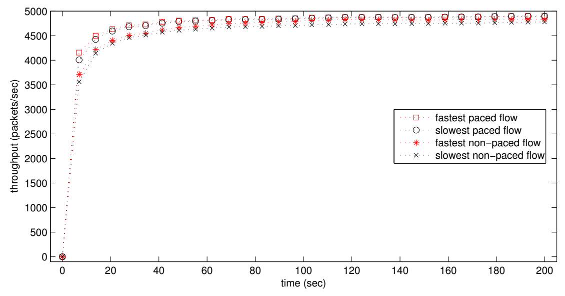

Fig. 5. [Scenario 2] The throughput of the slowest flow and the fastest flow among the 16 FAST flows in WAN.

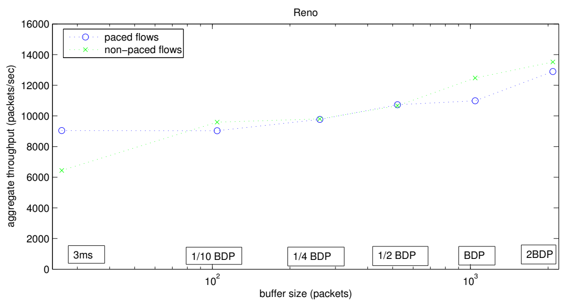

Fig. 6. [Scenario 3] The worst-flow throughput among 5 Reno flows in WAN.

burst will see a larger average queueing delay[5], due to the excessive queueing delay introduced by burstiness. This result is consistent with the experimental result reported in [14].

With pacing, such effect is eliminated and the fastest paced flow and slowest paced flow have the same throughput.

4) Scenario 3: worst-flow throughput with different buffer sizes.: In this scenario, we vary the buffer size in the bottle-neck router and see how the protocols' worst-flow throughput change in longer flows.

We use the same scenario as in Scenario 2, with the change of bottleneck buffer sizes. The RTT of Reno is scaled down 12ms since Reno cannot finish one congestion epoch in 200 seconds if the RTT is 120ms.

Figure 6, and Figure 7 show the throughput of the fastest flows and the throughput of the slowest flows with Reno

[5]FAST uses the average queueing delay over one window as the congestion signal.

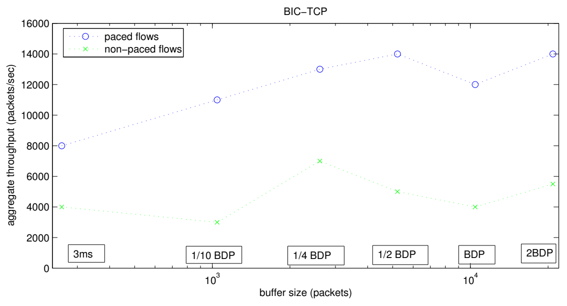

Fig. 7. [Scenario 3] The worst-flow throughput among 5 BIC-TCP flows in WAN.

and BIC-TCP.[6] Each point in the figure corresponds to the throughput measured at 200 second in a simulation.

We can see that Reno converges to fairness after 200 seconds, except when the buffer size is extremely small.

BIC-TCP does not converge to fairness even after 200 seconds. And the unfairness persists with different bottleneck buffer sizes. With pacing, BIC-TCP can double its worst-flow throughput in most cases.

Hence, in terms of maximum latency or worst-flow throughput, pacing can greatly improve the performance of BIC-TCP in both short term and long term.

C. *Performance: Aggregate Throughput*

A very important observation in [6] is that the aggregate throughput of paced TCP may be higher or lower than the non-paced TCP due to synchronization effects. In this section, we calculate an upper bound of throughput loss due to synchronization. Our upper bound shows that the synchronization effect becomes less significant to the aggregate throughput as the new TCP congestion control algorithms are applied.

1) Loss of aggregate throughput due to synchronization: Our calculation is inspired by the analysis in [8]. [8] points out that synchronization has a negative effect on aggregate throughput of TCP. The worst case happens when all the flows are synchronized. A direct application of equation (7) in [8] shows that Reno's throughput in the worst case (fully synchronized flows sharing a bottleneck with infinitely small buffer) is 75% of the capacity.[7] Hence, the throughput loss due to synchronization is 25%. This gives an upper bound of throughput loss due to synchronization for Reno with infinitely small buffer.

We give a simple estimation method for different TCP congestion control algorithm.

Given N homogeneous sources with round trip propagation delay of d sharing a single bottleneck with buffers size of $B * N$ and capacity of $c * N$. In the worst case, all the flows are synchronized. Each flow's behavior is equivalent to a single TCP flow using a bottleneck with capacity of c and buffer size of B. Hence we can estimate the aggregate throughput of N synchronized flows by the throughput of a single TCP flow.

We use T to denote the number of round trip for a flow to finish a congestion epoch.

In one congestion epoch, we use T_1 to denote the number of round trip for a flow to recover to full bandwidth utilization. We use P_1 to denote the number of packets transmitted in this period of time and P_2 to denote the number of packets transmitted in the rest of the congestion epoch.

The number of packets transmitted in one congestion epoch is $P_1 + P_2$.

In the first T_1 RTTs, the flow under-utilizes the bottleneck capacity. Hence, the number of packets can be sent in one congestion epoch, with full bottleneck utilization, is $cdT_1 + P_2$.

The throughput loss is $1 - \frac{P_1+P_2}{cdT_1+P_2}$.

[6]FAST has almost the same worst-flow throughput with all the buffer sizes.

[7]Let $p = 1$ in (7) of [8], we have $E\left(X^{(i)}\right) = \frac{C}{2N}$. This is the throughput after rate halving. Hence the average throughput over the whole congestion epoch is $\frac{E\left(X^{(i)}\right)+\frac{C}{N}}{2} = \frac{3}{4}\frac{C}{N}$.

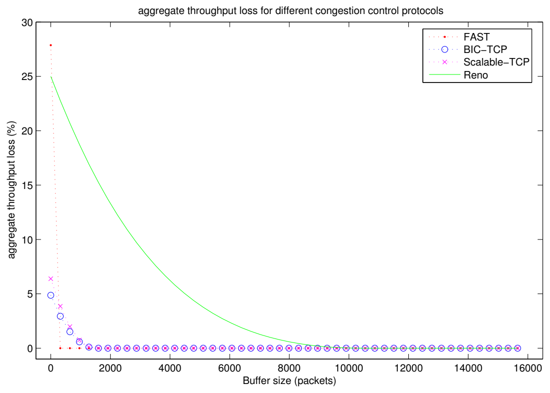

Fig. 8. Synchronization throughput loss of different congestion control algorithm (Bandwidth Delay Product = 10440 packets)

Take Reno as an example. Let the maximum window size for a flow to be w_{max} where $w_{max} = B + cd$. After one loss event, the congestion window is reduced to $\frac{1}{2}w_{max}$. It takes $\frac{1}{2}w_{max}$ RTT to recover to w_{max} and generate another loss. Hence the length of congestion epoch is $T = \frac{1}{2}w_{max}$ RTT.

The flow fully utilizes the link after the window size reaches cd. Hence

$$T_1 = cd - \frac{1}{2}w_{max} = \frac{1}{2}(cd - B)$$

and

$$P_1 = \frac{\frac{1}{2}w_{max} + cd}{2}T_1 = \frac{1}{8}B^2 + \frac{3}{8}(cd)^2 - \frac{1}{4}cdB$$

and

$$P_2 = \frac{cd + w_{max}}{2}(T - T_1) = cdB + \frac{1}{2}B^2$$

Hence the throughput loss due to synchronization is $\frac{1}{4}\frac{(cd)^2+cdB-B^2}{(cd)^2+cdB+B^2}$. When $B = 0$, we get the result of 25% as we get from [8].

Similarly, we can calculate the throughput loss due to synchronization for other congestion algorithms such as Scalable TCP, BIC-TCP and FAST. Figure 8 is the calculation result for these congestion control algorithms, with different buffer size, under 1Gbps link, 120ms round trip propagation delay and standard packet size (MTU=1500). We can see that all the new congestion control algorithms have much smaller throughput loss when the loss signals are synchronized.[8] Working with these new congestion control algorithms, the synchronization of pacing will be much less significant.

We simulate scenarios with capacity of 1Gbps, different TCP variants and different buffer sizes. Our results show that paced TCP flows has better performance than non-paced flows when buffer size is very small, due to its advantages in lower level control. When buffer size is moderate and

[8]Note that FAST has zero throughput loss except a point where buffer is extremely small. When buffer size is very small, the multiplicative increment scheme in FAST is dominant and FAST works as an MIMD algorithm. When buffer size is large enough to hold the target number of packets, FAST does not depend on packet loss events and its throughput does not suffer from synchronization. Actually, the design of FAST is based on synchronization assumption (See [23]).

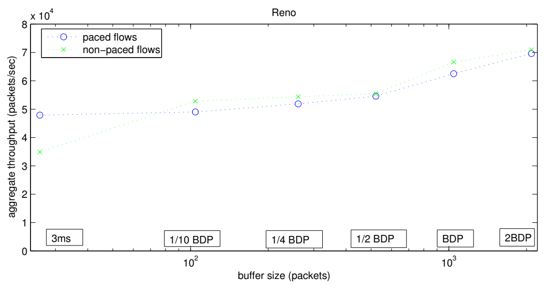

Fig. 9. [Scenario 4] The aggregate throughput of 5 Reno flows in WAN.

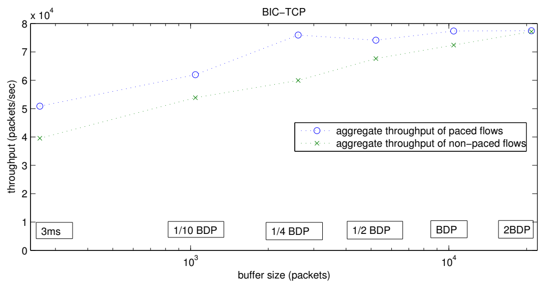

Fig. 10. [Scenario 4] The aggregate throughput of 5 BIC-TCP flows in WAN.

large, pacing has similar throughput to non-paced flows since pacing's synchronization effect has less significant impact on the aggregate throughput.

2) *Scenario 4: Aggregate throughput of different TCP variants:* In this scenario, we run five TCP flows sharing a bottleneck with 1Gbps and varies the buffer size in the bottleneck. The access link has a capacity of 1Gbps and enough buffer for bandwidth delay product packets. The delay of the flows are 120ms (with Reno's RTT downscale to 12ms). We compare the aggregate throughput when the TCP algorithm is Reno, BIC-TCP or FAST.

Figure 9, Figure 10 and Figure 11 shows the aggregate throughput of Reno, BIC-TCP and FAST correspondingly. From Reno to BIC-TCP to FAST, the throughput of paced TCP becomes better and better, as predicted by our calculation.

Although paced Reno slightly loses to non-paced Reno due to synchronization, the performance loss is within the upper bound of 25% in our prediction.

One interesting phenomenon is that pacing can improve the performance of BIC-TCP even when the buffer size is equal to bandwidth delay product. This is different from the common

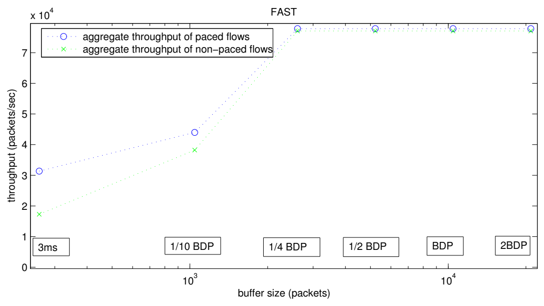

Fig. 11. [Scenario 4] The aggregate throughput of 5 FAST flows in WAN.

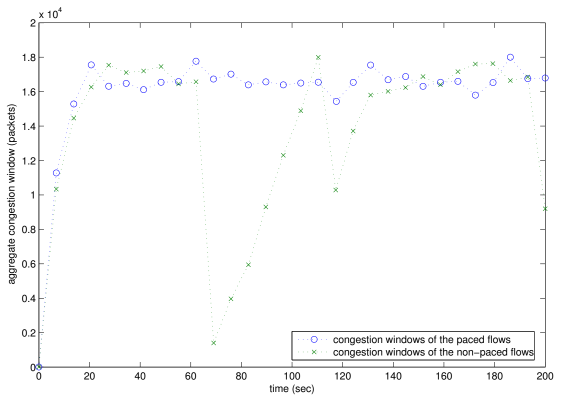

Fig. 12. [Scenario 4] The aggregate window of 5 BIC-TCP flows.

observation on Reno (e.g. in [6]). We varied the number of flows from 2 to 16 and got the same observation. We zoom in this case and Figure 12 shows the aggregate congestion window of the non-paced flows. From this figure, we can see there are several huge drops of congestion window in non-paced flows. These correspond to the moments when some BIC-TCP flows enter *BIC slow start* where their congestion windows are increased exponentially. If there are multiple BIC-TCP flows enter *BIC slow start* in the same RTT, they generate very bursty traffic and multiple packet loss events, which force several BIC-TCP flows eventually timeout. This phenomenon is not only associated with BIC. We have similar observation on HS-TCP, Scalable TCP and FAST. These high speed TCP variants increase their congestion windows more aggressively and hence generate more bursty traffic in long run. With these TCP variants, pacing's advantage in sub-RTT level is important not only in startup phase but also in the whole life of a flow.

We run the same scenarios with heterogeneous sources (flows with different propagation round trip time). We get similar conclusions as in the scenarios with homogeneous sources.

III. Migration from non-paced world to paced world

In this section, we study the performance of paced and nonpaced flows when they share the same network, and discuss its implications on migration from a nonpaced world to a paced world.

A. Improving the performance of pacing for mixed environment

[6] observed that paced Reno flows lose out to non-paced Reno flows in competition. The common explanation for the competition results is that non-paced flows transmit in bursts and have a smaller probability of experiencing a drop as opposed to paced flows which spread their packets uniformly in the whole RTT. This implies that non-paced flows should see a smaller loss event rate than paced flows and hence have higher throughput.

Our results, however, show that non-paced flows experience *more* loss events than paced flows, in most of the scenarios, while they still win over the paced flows. Instead of the difference in loss event rates, a straight-forward implementation of pacing, which paces out the packets at intervals of $\frac{\text{RTT}}{\text{congestion window}}$, contributes largely to the performance degradation. [9]

Most of the TCP implementations (e.g. Linux, NS-2 and etc) rely on the acknowledgment arrival events. The congestion window is increased by either 1 (in slow start phase) or by $\frac{1}{\text{congestion window}}$ (in congestion avoidance phase) when an acknowledgment is received. Hence, the congestion window is changing within one RTT. If pacing simply paces packets at intervals of $\frac{\text{RTT}}{\text{congestion window}}$ and if the congestion window is increasing, the number of packets transmitted in one RTT is smaller than the congestion window size at the end of the RTT. That is, fewer packets are sent out in one RTT than expected. In return, less acknowledgments will come back in the next RTT and the congestion window is growing slower.

To illustrate this effect, we give an example as below. Assume one flow is in its slow start phase. Its congestion window is 4 at the beginning of an RTT and the congestion window is increased to 8 at the end of that RTT. 8 packets should be sent out in this RTT. We label them $p_1, p_2 \cdots p_8$ and their transmission times $t_1, t_2, \cdots t_8$ where we let $t_1 = 0$. If we pace the packets with interval of $\frac{\text{RTT}}{\text{congestion window}}$, the first two packets are sent out with a congestion window of 5. They have an interval of $\frac{1}{5}$RTT. That is $t_2 = \frac{1}{5}$RTT. Similarly, the interval between p_2 and p_3 is $\frac{1}{6}$RTT, as the interval between p_3 and p_4. Hence we have $t_3 = t_2 + \frac{1}{6}$RTT and $t_4 = t_3 + \frac{1}{6}$RTT. With the same calculation, we can get $t_5 = t_4 + \frac{1}{7}$RTT, $t_6 = t_5 + \frac{1}{7}$RTT, $t_7 = t_6 + \frac{1}{8}$RTT, and $t_8 = t_7 + \frac{1}{8}$RTT.

That is,

$$t_8 = 2\sum_{i=5}^{i=8} \frac{1}{i}\text{RTT} = 1.3\text{RTT} \tag{6}$$

That means, p_8 is sent in the next RTT while the correct time of sending p_8 should be the end of this RTT.

This problem contributes significantly to the case where paced Reno loses to non-paced Reno since the congestion windows of the paced flows grow much slower than the congestion windows of the non-paced flows do.

This problem can be fixed by using the congestion window for the next RTT in pacing. In congestion avoidance phase, this means $\text{cwnd} + 1$. In slow start phase, this means $\min\{2 * \text{cwnd}\ , \text{SSthreshold}\}$. Hence, instead of using $\frac{\text{RTT}}{\text{cwnd}}$ to pace packets in pacing, we should use the expected congestion window in next RTT to pace the packets, which is

$$\frac{\text{RTT}}{\max\{\text{cwnd} + 1, \min\{2 * \text{cwnd}\ , \text{SSthreshold}\}\}} \tag{7}$$

[9]This problem also hurts the performance of pacing in non-competition situations. But in non-competition cases, its effect only appears in the form of slightly larger loss of aggregate throughput due to synchronization. In the competition cases, its effect let non-paced flows to have larger congestion windows than paced flows. The difference in congestion windows result in persistent and significant performance difference.

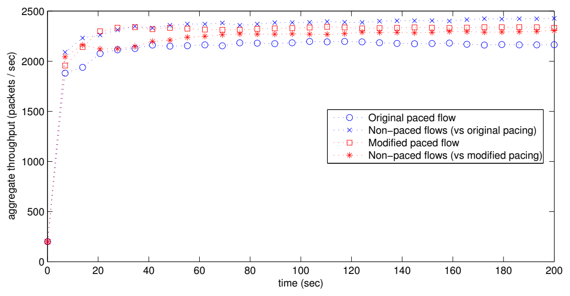

Fig. 13. [Scenario 5] paced flows vs non-paced flows, when Reno is used

	paced	non-paced
per-packet loss rate	0.6469%	0.8663%
loss event rate	6.75%	8.75%

TABLE I

LOSS RATES OF ORIGINAL PACED RENO AND NON-PACED RENO IN COMPETITION

With this improvement, paced Reno no longer loses to non-paced Reno. In the later part of the paper, we use “modified pacing” to refer the pacing algorithm with this improvement and “original pacing” to refer the pacing algorithm without this improvement.

The following scenario shows the effectiveness of this imrpovement.

B. Scenario 5: Improvement of pacing for mixed environment

In this scenario, 20 paced flows and 20 non-paced flows are sharing a bottleneck of 57Mbps, with a round trip propagation delay of 100 ms. This is similar to the scenarios in [6]. We show the aggregate rate over all the paced flows and the aggregate rate over all the non-paced flows in the life of the simulation.

If we use the original pacing in Reno, we observe the same phenomenon that paced flows loss out to non-paced flows, as shown by the blue curves in Figure 13. We measure the average per-packet loss rate and average loss event rate[10]of the flows in Table I. The loss event rate of non-paced flows are higher than the paced flows, which means that loss event rate is not the source of paced flows’ performance problem.

With the modified pacing algorithm, paced flows have the same aggregate throughput as the non-paced flows, as shown by the red curves in Figure 13.

Simulations with New-Reno show the same conclusion.

However, when working with SACK, the modified paced flows still lose out to non-paced flows, though the difference is much smaller than the one with the original pacing. Figure 14 shows shows the results with TCP SACK, with the same scenario. In other protocols that utilize SACK in loss recovery (e.g. FACK, BIC-TCP and etc), paced flows lose out to non-paced flows.

[10]Loss event rate is calculated as the number of RTTs experiencing loss over the number of RTTs in the life of a flow. The average is taken over all the paced flows or non-paced flows

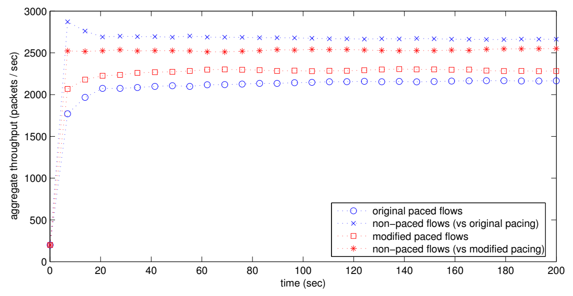

Fig. 14. [Scenario 5] paced flows vs non-paced flows, when Reno is used

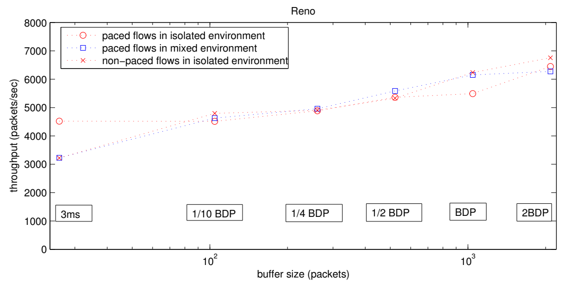

Fig. 15. [Scenario 6] Paced flows in mixed environment, worst-flow throughput of Reno

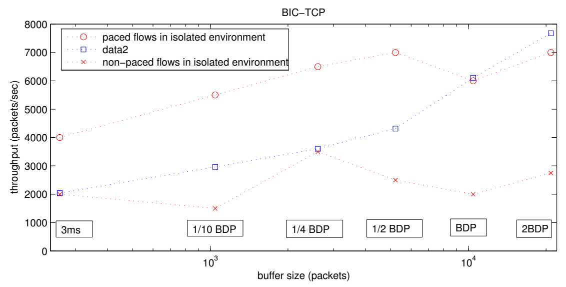

Fig. 16. [Scenario 6] Paced flows in mixed environment, worst-flow throughput of BIC

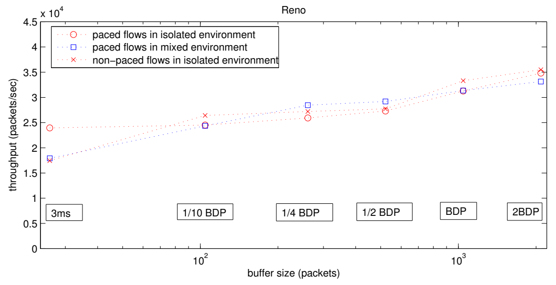

Fig. 17. [Scenario 7] Paced flows in mixed environment, aggregate throughput of Reno

It is not clear why paced flows with SACK lose out to non-paced SACK flows. One potential problem is that SACK favors the loss patterns of non-paced flows. When there are lost packets in one window, non-paced flows lose their packets in bursts and paced flows' lost packets scatter within the whole window. Recovering a burst of packets only needs one SACK block, no matter how large the burst is. So non-paced senders can quickly exit loss recovery state while paced senders have to wait for more SACKs before they can exit loss recovery state and grow their windows.

In the following two subsections, we re-exam the worst-flow throughput and aggregate throughput of the paced flows, in an environment where 50% of the flows are paced (with the modified pacing algorithm) and 50% of the flows are non-paced. We compare the results with the performance in isolated environment. In most of our simulation results, paced TCP in mixed environment still has better performance than non-paced TCP in isolation, though paced TCP works better in isolation environment.

C. Scenario 6: Worst-flow throughput in mixed environment

In this subsection, we study worst-flow throughput of pacing, in mixed environment.

We repeat the same scenario as in Scenario 3 of Section II-B except that we have 50% paced flows and 50% non-paced flows.

Figure 15 and 16 show the worst-flow throughput of Reno and BIC. [11]

When the buffer size is small, paced flows perform worse than they do in the isolated environment, since the paced flows in mixed environment also observe the queueing delay and packet loss events introduced by non-paced flows. As the buffer size becomes large, the performance of paced flows in mixed environment is similar to their performance in isolated environment. In these cases, there are much less loss events and the effect of SACK is not significant. Other advantages of paced flows dominate.

In most of the cases, paced flows in mixed environment are still performing better than non-paced flows in isolation. Hence, if only a subset of the flows adapt pacing, they still have better performance than if they do not pace.

D. Scenario 7: Aggregate throughput in mixed environment

In this subsection, we exam the aggregate throughput. We repeat the same scenario as in Scenario 4 of Section II-C.2 except that we have 50% paced flows and 50% non-paced flows.

Figure 17 , Figure 18 and Figure 19

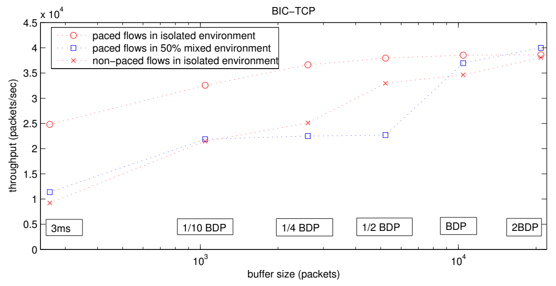

Fig. 18. [Scenario 7] Paced flows in mixed environment, aggregate throughput of BIC

[11]Again, FAST has very similar worst-flow throughput when it is paced or non-paced.

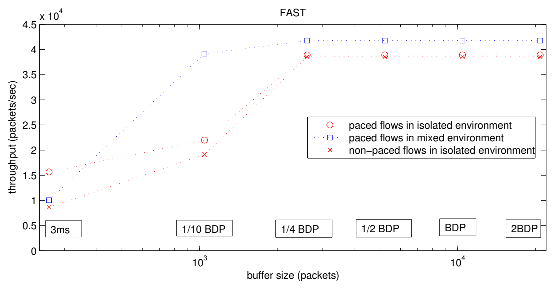

Fig. 19. [Scenario 7] Paced flows in mixed environment, aggregate throughput of FAST

show the aggregate throughput of Reno, BIC-TCP and FAST.

In terms of aggregate throughput, we see the same trend that paced flows perform better when buffer size is large. We believe that SACK effect is again dominant in the cases with small buffers.

When the buffer size is large enough to hold FAST's α packets, FAST flows have higher aggregate throughput than what they have in isolated environment. With pacing, a FAST flow experiences less noise in queueing delay introduced by its own burstiness. It sees a smaller average queueing delay than a non-paced flow sees, and grabs some bandwidth from the non-paced flows in competition.

From Figure 15 and Figure 17, we see no clear advantage for paced flows in mixed environments. Hence, Reno users may not have the motivation to adapt pacing.

Figure 18 shows that BIC-TCP loses some aggregate bandwidth in certain cases when it changes from non-pacing to pacing in mixed environment. This tells that a BIC-TCP user may not want to adapt pacing if (s)he tries to optimize the aggregate throughput.

On the other hand, Figure 16 shows that BIC-TCP users have a strong motivation to adapt pacing if worst-flow throughput is their major concerns. And Figure 19 shows that FAST users should always use pacing.

Hence, the motivation of using pacing depends on the TCP variants and the performance metrics an application cares. As we are moving towards the high speed TCP protocols, the motivation gets stronger.

E. Mechanisms related to the performance of pacing

In this subsection, we summarize the TCP mechanisms that affect the performance of pacing. There are at least three factors that have significant effects on performance of pacing:

1) Smoothness of packet transmission in sub-RTT level control: This is the original motivation of pacing. Pacing can help to reduce excessive loss or delay introduced by ack-clocking. Also, by smoothening the packet transmission, pacing is robust to the ack-compression and stretch-ack.

This has positive effect on pacing with all the TCP implementations, from original Reno to new variants such as the loss-based BIC-TCP and delay-based FAST.

2) Synchronization in congestion signals: As pointed out by [6], synchronization has two effects on performance of TCP: it degrades the aggregate throughput but improves the fairness. Pacing's synchronization in slow start phase has a negative effect on its aggregate throughput and a positive effect on its fairness.

The effect of synchronization on aggregate throughput becomes less and less significant as the TCP congestion control evolves from Reno to BIC-TCP and FAST. The single flow performance of the new TCP variants are all much better than Reno. Even all flows are synchronize and the aggregate behavior is equivalent to a single flow, the loss of aggregate throughput is insignificant.

However, the effect of synchronization on fairness is still important for all the loss-based congestion control algorithms. Most of the loss-based congestion control algorithms take many RTTs to converge to fairness, especially in high speed long distance network. Synchronization can help to improve the fairness in startup phase and greatly shorten the convergence time.

3) Loss recovery: Finally, the loss recovery mechanisms have an impact on the performance of pacing. For example, SACK favors non-paced flows since a single SACK can recover a burst of packets but not several packets scattering within a window. With SACK, the performance of non-paced flows is greatly improved but the the paced flows get less benefit. This is a relatively negative effect for paced flows in mixed environment.

All the loss-based congestion control algorithms are still heavily rely on SACK.

We listed in these factors in Table II and see how their significance changes as TCP variants evolves.

The first two rows are the factors that have positive effect on pacing's performance. They still have significant importance as TCP evolves from Reno to SACK to BIC-TCP to FAST (except fairness of FAST).

The last two rows are the factors that have negative effect on on pacing's performance. Synchronization effect on aggregate throughput becomes less significant, while SACK effect remains significant for loss-based congestion control algorithms.

In the future, if the loss recovery mechanism is more powerful to recover lost packets in multiple locations, there will be a stronger motivation for adapting pacing universally.

IV. Conclusion and Open problems

TCP Pacing has a multi-facet impact on flow performance. Pacing clearly improves performance in terms of "worst-flow latency", when the routers are under-buffered, and when new high-speed TCP variants are used. However, it fragments the packet losses, leads to more SACK blocks needed to convey loss information, and does not obtain its fair share of the bandwidth when competing with non-paced flows. However, in many cases, pacing improved both the throughput of paced flows and that of non-paced flows, offering an incentive for applications to use pacing.

In short, for the application programmers, we would like to send the message that pacing is good, particularly if the application uses multiple concurrent TCP flows to accomplish a task. For the high-performance computing community, we

	Reno	SACK	BIC-TCP	FAST
Smoothness of packet transmission	Important	Important	Important	Important
Synchronization in congestion signal on fairness	Important	Important	Important	Not Important
Synchronization in congestion signal on aggregate throughput	Important	Important	Not Important	Not Important
Loss recovery	n/a	Important	Important	Not Important

TABLE II

THE EFFECT OF DIFFERENT TCP MECHANISMS ON DIFFERENT TCP VARIANTS

also want to encourage the use of pacing, as it improves the performance of new high-speed TCP protocols in the presence of under-buffered routers and switches. To the networking research community, our message is that the interaction between pacing, packet loss patterns and TCP protocol mechanisms such as SACK is a rich area that deserves more exploration.

REFERENCES

[1] BIC-TCP website. URL: http://www.csc.ncsu.edu/faculty/rhee/export/bitcp.

[2] DataTAG. URL: http://datatag.web.cern.ch.

[3] FAST TCP simulator module for ns-2. URL: http://www.cubinlab.ee.mu.oz.au/ns2fasttcp/.

[4] Sybase inc. URL: http://www.sybase.com/.

[5] xrootd. URL: http://xrootd.slac.stanford.edu/.

[6] AGGARWAL, A., SAVAGE, S., AND ANDERSON, T. Understanding the performance of TCP pacing. In *Proceedings on IEEE Infocom 2000* (2000), pp. 1157–1165.

[7] APPENZELLER, G., KESLASSY, I., AND MCKEOWN, N. Sizing router buffers. In *Proceedings of ACM SIGCOMM '04, August /September 2004* (2004).

[8] BACCELLI, F., AND HONG, D. AIMD, fairness and fractal scaling of TCP traffic. In *Proceedings on IEEE Infocom 2002* (2002).

[9] CHIU, D., AND JAIN, R. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks 17* (1989), 1–14.

[10] FLOYD, S. Highspeed tcp for large congestion windows, 2002.

[11] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. The google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2003), ACM Press, pp. 29–43.

[12] HOE, J. C. Improving the start-up behavior of a congestion control scheme for tcp. In *Conference proceedings on Applications, technologies, architectures, and protocols for computer communications* (1996), ACM Press, pp. 270–280.

[13] INC., P. Controlling WAN bandwdith and application traffic. http://www.packeteer.com/resources/prod-sol/ControlDrillDown.pdf.

[14] JIN, C., WEI, D. X., AND LOW, S. H. TCP FAST: motivation, architecture, algorithms, performance. Proceedings of IEEE Infocom 2004, July 2003.

[15] KAMEZAWA, H., NAKAMURA, M., TAMATSUKURI, J., AOSHIMA, N., INABA, M., AND HIRAKI, K. Inter-layer coordination for parallel tcp streams on long fat pipe networks. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing* (Washington, DC, USA, 2004), IEEE Computer Society, p. 24.

[16] KAMEZAWA, H., NAKAMURA, M., TAMATSUKURI, J., AOSHIMA, N., INABA, M., HIRAKI, K., SHITAMI, J., JINZAKI, A., KURUSU, R., AND IKUTA, M. S. Y. Inter-layer coordination for parallel TCP streams on long fat pipe networks. In *Proceedings of SuperComputing 2004* (2004).

[17] KULIK, J., COUTLER, R., ROCKWELL, D., AND PARTRIDGE, C. A simulation study of paced TCP. Tech. Rep. BBN Technical Memorandum No. 1218, BBN Technologies, 1999.

[18] LI, Y.-T., LEITH, D., AND SHORTEN, R. N. Experimental evaluation of tcp protocols for high-speed networks. http://hamilton.ie/net/eval/HI2005.htm.

[19] MATHIS, M., AND MAHDAVI, J. Forward acknowledgement: refining tcp congestion control. In *Conference proceedings on Applications, technologies, architectures, and protocols for computer communications* (1996), ACM Press, pp. 281–291.

[20] RHEE, I., AND XU, L. Limitations of equation-based congestion control. In *Proceedings of ACM Sigcomm 2005* (2005).

[21] UNIVERSITY OF TOKYO. Data reservoir. http://data-reservoir.adm.s.u-tokyo.ac.jp/.

[22] VISWESWARAIAH, V., AND HEIDEMANN, J. Improving restart of idle TCP connections. submitted for publication, July 1997.

[23] WANG, J., WEI, D., AND LOW, S. Modeling and stability of fast tcp. In *Proceedings of IEEE Infocom 2005* (2005).

[24] WEI, D., HEDGE, S., AND LOW, S. A burstiness control for TCP. In *Proceedings of PFLDNet 2005* (2005).

[25] XU, L., HARFOUSH, K., AND RHEE, I. Binary increase congestion control for fast long-distance networks. In *Proceedings of IEEE Infocom 2004* (2004).

[26] ZHANG, L., SHENKER, S., AND CLARK, D. D. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. In *Proceedings of the ACM SIGCOMM 1991 Conference on CommunicationsArchitectures and Protocols* (1991), pp. 133–147.