Paced Chirping: Rapid flow start with very low queuing delay

Joakim Misund
University of Oslo, Department of Informatics

Bob Briscoe
Independent

Abstract—**This paper introduces a promising new direction for getting a traffic flow up to speed fast while keeping the maximum queuing delay that the new flow adds extremely low. It is therefore most interesting in environments where queue delay is already fairly low. Nonetheless, it requires no special network infrastructure, being solely delay-based, so it ought to be applicable to the general Internet.**

Received wisdom from TCP slow-start is that the faster a flow accelerates, the more it will overshoot the queue before the sender will notice one round trip later. The proposed technique, called paced chirping. escapes that dilemma. The sender pulses the queue increasingly rapidly with trains of packets called 'chirps' that it crafts to rapidly estimate available capacity. Critically, the sender relaxes the queue between chirps, so the queue never accumulates more than a few packets. Thus, paced chirping escapes the overshoot dilemma, but still pushes enough against any pre-existing flows, so they yield their capacity.

The algorithm has been implemented in Linux and shows great promise from initial evaluation. Work so far has set aside numerous issues that are all important, but not central to proving the concept, e.g. handling delayed ACKs, losses, ECN marking, reordering, variable rate links, etc, The work and the code is being published at this stage to seek review and collaboration around this promising direction.

I. Introduction

This paper introduces a promising new direction for getting a traffic flow up to speed fast while keeping the maximum delay that the new flow adds extremely low (2–3 ms over a typical Internet RTT of 20 ms).

It is therefore most interesting in environments where a congestion control like Data Centre TCP (DCTCP [2]) already keeps queuing delay extremely low. This might be in a data centre itself, or over the Internet within the L4S architecture being standardized at the IETF [4], [6].

The testbed experiments reported in this paper assume an L4S environment within the Internet. Nonetheless, the solution requires no special network infrastructure, being solely delay-based. Therefore we have open-sourced the code to encourage others to try it within other end-to-end transports and over other networks.

Received wisdom from TCP slow-start is that the faster a flow accelerates, the more it will overshoot and cause queuing delay before the sender will be able to notice one round trip later. The proposed technique escapes that dilemma.

The sender pulses the bottleneck queue with 'chirps' that are short trains of packets sent increasingly close together [19]. The sender can rapidly estimate the available capacity using the relative delay of acknowledgements. To minimize delay to itself and others, the sender prevents the queue accumulating any more than a few packets by allowing a guard interval between each chirp so the queue can relax. It uses the variability of its estimates to adapt the guard interval. This approach was informed by our experience with TCP RAPID [14], which doubles its window as in the traditional slow-start and only uses chirps to determine when to exit slow start. It also starts each chirp directly after the previous one, so an overestimate can still overshoot for a whole round.

With paced chirping, each chirp still causes competing traffic to make space for the new flow. It tracks this increasing availability of capacity and keeps pushing against it, both by spacing the packets within a chirp more closely and by spacing the whole chirps more closely—hence the name 'paced chirping'.

The paced chirping algorithm has been implemented in Linux and shows great promise from initial evaluation. This work solely focuses on proving the concept. If that survives critical review, there is much further work to do (listed at the end). The most critical being the need to be able to control delayed ACKs from the sender and the need to tune and evaluate over variable rate links.

A. The Problem

Cutting delay in communications is a multifaceted problem [5]. Most modern applications are latency-sensitive; not just the usual examples, voice and gaming, but also instant messaging and most uses of the web, as well as remote desktop, interactive apps based in the cloud and real-time apps such as conversational / interactive video, augmented reality and remote control of industrial processes.

For virtual reality to feel natural, as a rule of thumb the lag has to stay below about 20 ms end-to-end [1], [7]. The speed of light equates every millisecond of round-trip delay to 100 km in glass (or 150 km in air). So, if delays other than propagation could be reduced from say 15ms to 1ms, for the same natural perceptual experience, servers would only have to be distributed to within 1,900 km (19 ms) of each user, rather than within 500 km (5 ms).

Particularly for real-time apps, it is important to cut the high delays in the tail of the latency distribution, not just reduce the average. By far the greatest cause of delay variation is queuing. Modern active queue management (AQM) can reduce the average delay under load to 5–15 ms [13].

However, the 99th percentile (P99) queue can still reach 20–100 ms depending on load. Per-flow queuing isolates one flow from the queuing of another. However, amongst other problems, it does not protect a flow from its own delay variation, which is important for emerging interactive real-time apps that need both low delay and high throughput.

Data centre TCP (DCTCP) proves that flows can start up with very little extra queuing delay. And the L4S approach being standardized by the IETF [4], [6] makes it possible and safe to deploy so-called 'scalable' congestion controls derived from data centre TCP (DCTCP) over the public Internet. Thus, L4S seems to solve the problem, cutting P99 queuing over the public Internet to about 1 ms [8]. However, only at the expense of greatly increased convergence time.

In a data centre environment, the convergence problem is caused by two factors: i) new flows exit slow start early because of the higher prevailing ECN marking probability induced by established DCTCP flows and ii) established DCTCP flows respond 1.5 to 2× more slowly than TCP [3].

[8] achieved ultra-low queuing delay in an Internet environment using the Linux DCTCP code that supports mixed round trip times [3] over an L4S DualQ Coupled AQM [9]. We use the same arrangement in our experiments and show that DCTCP's convergence problem is a lot worse than in a data centre (see Figure 1). The problem is compounded by the greater typical disparity between the line rate of the sender and of the bottleneck. While the sender is probing for the bottleneck rate, it is hard to avoid tripping over the shallow ECN-marking threshold and exiting slow-start early.

Currently, the paced chirping solution gathers ECN feedback to build a moving average level of congestion, but we have temporarily suppressed any response to ECN until the congestion avoidance phase starts. We have plans for paced chirping to improve its precision if ECN signals are available, but it will respond to their extent, not just the existence of a single mark.

Paced chirping deliberately depends primarily on delay measurements because, when it starts, a flow cannot be sure whether the bottleneck supports L4S-ECN. Usually, if the sender receives ECN feedback after very little growth in queue delay, it can assume an L4S-ECN bottleneck. However, when cross-traffic at the bottleneck is yielding capacity to a rapidly growing flow, we have to allow for a new node becoming the bottleneck, which we cannot assume will support L4S.

B. *Contributions*

The main contribution of this paper is an algorithm that should be able to keep the additional queue at flow-start scale-independent (and it does in our limited experiments). This is achieved by leaving a guard interval between each chirp, and reducing it more rapidly, the more consistent the repeated measurements of available capacity are. Our understanding of how this adaptation should proceed is still developing, but even with no adaption, the algorithm performs extremely well at fixed capacity bottlenecks.

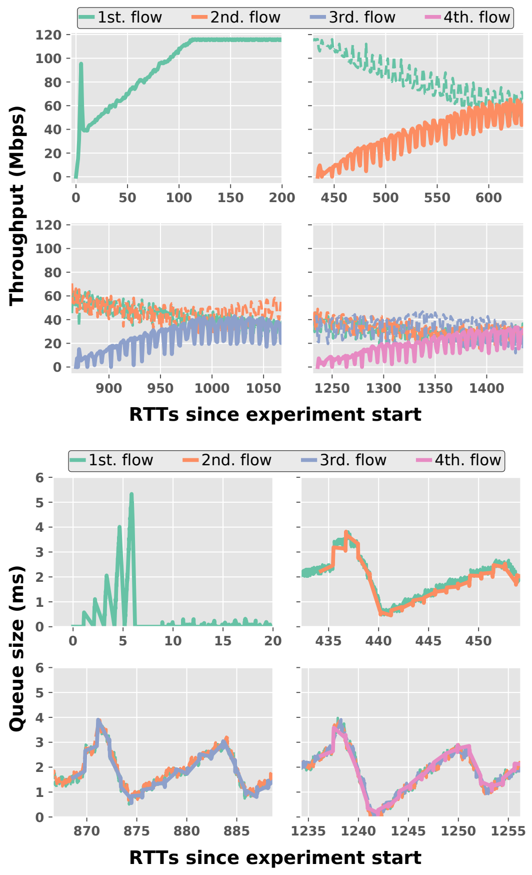


Fig. 1. Throughput and Queue delay of 4 DCTCP flows in an Internet environment starting one after another. Each new flow hardly increases queuing delay at all, but each flow exits slow-start very early making convergence to steady state very slow. ECN-marking threshold 0.17*BDP; RTT 15 ms; capacity 120 Mb/s.

The wider contribution is an approach that introduces closed-loop control after the first round trip. it continually crafts the spacing between sent packets around measurements exploiting the patterns used in previous rounds, while allowing for the possibility of change and error.

We have also contributed our Linux code as open-source. It includes a general facility in the kernel for releasing packets at a list of times (similar to timing wheels [20]), based on internal pacing. We have also contributed initial evaluations of paced chirping in an L4S environment, including comparisons with hybrid slow-start and DCTCP.

II. Solution Design

The approach uses existing work on packet chirping to estimate the available bottleneck capacity of the network path [19], briefly outlined below. Loosely, the sender sets the average rate of each chirp to its estimate of available capacity from previous rounds. However, we pace the start of each chirp to introduce a guard interval to allow for estimation error, as also outlined below.

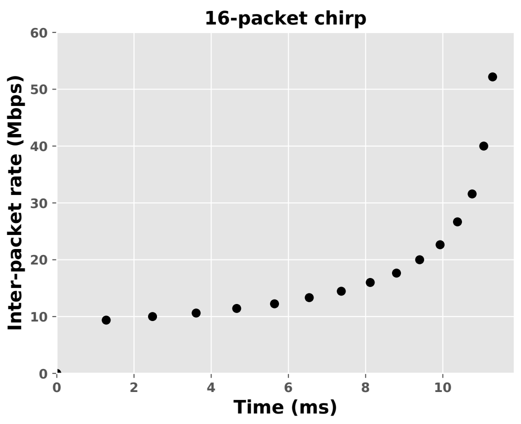


Fig. 2. Packets in a chirp are sent with decreasing inter-packet interval (increasing inter-packet rate).

A. Chirping

A flow start mechanism needs to know available capacity of the bottleneck link, not the maximum capacity. A data sender can measure available capacity with packet chirps, which are sequences of packets sent with decreasing inter-packet interval. Figure 2 shows the inter-packet rates in a 16-packet chirp. The sender measures available capacity by detecting when the inter-packet delay at the receiver flattens off. It achieves this by comparing the inter-packet intervals of the sent and received packets. Ideally one-way delay would be used but round-trip delay using the TCP timestamp option is more convenient and it is sufficiently accurate in the absence of reverse path congestion.

The processing cost can be kept low by using an arithmetic series for the gaps [15]. We use microsecond precision timers. Nonetheless, chirping is inherently robust to the actual time a packet is released as long as the gaps tend to reduce. We currently use the techniques from pathchirp [19] to filter out noise.

The alternative of measuring the ACK rate from a line-rate burst would not be useful. It would measure maximum capacity as if there were no background traffic, because the burst would squeeze numerous packets into the gaps between the background packets then measure their rate as the queue drains into the link.

Similarly, the alternative of sending at a constantly paced packet rate that increases each round [16] is not a fruitful way to resolve the acceleration vs. harm dilemma. Up until slightly below the available capacity, pacing causes minimal harm but gathers zero information about capacity. However, once the sender increases to even slightly above available capacity, the queue grows for the whole round trip, with no warning until a round trip later.

B. Paced Chirping

The goal is to create a close-loop flow-start algorithm from the start of the second round. Then the way flow-start

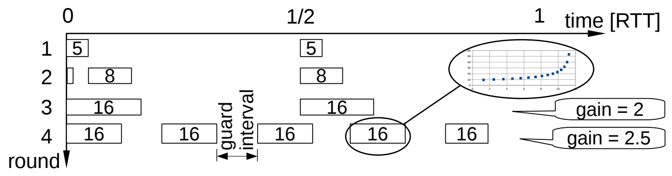


Fig. 3. The arrangement of chirps in each round as a flow starts up.

proceeds can depend on the run-time environment, not on arbitrary hard-coded design-time constants. It aims to maximize the ratio between the rate at which information is gathered about available capacity and the harm (queue delay) caused in the process. To this end, we believe there is considerable scope to further improve the algorithm illustrated in Figure 3 and outlined below or in detail in [18]. Nonetheless, it is already giving very promising results.

In the following two paragraphs we will first discuss how the average gap of each chirp is determined, and then discuss the 'guard interval' that the sender introduces to space out the chirps over the round in order to ensure the queue has time to relax.

As each round proceeds, the sender determines the average gap for the next chirp from an EWMA[1] of the gap estimates from previous rounds, where each gap estimate is the inter-packet gap at which queuing was found to start in each chirp. In Figure 3, each chirp is labelled with the number of packets within it. After round 3, the number no longer increases, but it can be seen that the duration of each chirp reduces. This is because the available capacity is increasing, due to background flows yielding in response to the congestion caused by the peaks of previous chirps.

If the gap estimate were perfect, even sending chirps back-to-back (as in TCP RAPID [14]) would allow time for the queue to relax. But the guard interval allows for error in the estimation. This avoids the queue ratcheting up, which aims both to avoid harm and to ensure that each chirp starts from a clear queue. By spreading out the times when samples of available capacity are taken we also hope to reduce the chance of completely missing other flows in the process of starting up.

Figure 3 illustrates the gain increasing from 2 to 2.5 between rounds 3 and 4 as the consistency of the measurements improve. The gain represents how much the window multiplies from one round to the next, and higher gain leads to a faster reduction in the guard interval.

This approach aims to ensure that the maximum queue delay solely depends on chirp geometry. Thus, queue delay will not depend on the ultimate window of packets per round trip. Therefore, as flow rates scale into the future, queue delay will remain scale-independent.

The spikes of each chirp ensure that some congestion occurs, so that any background flows are induced to yield

[1]We intend to weight each gap estimate to account for the noisiness of the measurements within the chirp it came from.

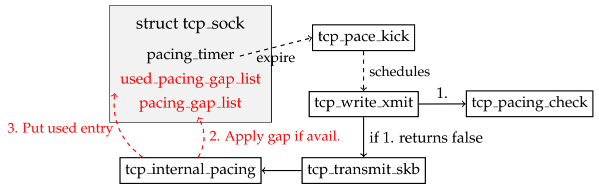


Fig. 4. Shows how tcp_internal_pacing interacts with the two list of packet gaps. If pacing_gap_list contains an entry the first entry is updated and its gap used to set the new pacing timer. The entry is then moved to used_pacing_gap_list for the CC modules use.

to a new flow[2]. As the sender introduces more chirps in each round, it induces more frequent congestion events, but with no greater pulse to the queue each time.

The more consistent recent capacity measurements are, the more the gain can increase. This is an area for futher experimentation, but even a naïve algorithm significantly reduced queue delay, albeit with slight loss in performance (see [18]). We have experimented with the guard interval reduce dependent on an EWMA of the variability of previous estimates. In this way, when the estimate repeats solidly (e.g. when the link is empty), the algorithm will accelerate rapidly to the available capacity but, when the estimate is noisy, the algorithm will accelerate more cautiously. We have also adapted the geometry of the chirp (see [18] for details).

Currently, we continue the paced chirping process until the chirps are back-to-back (no guard-interval) at which point the sender gracefully transitions to congestion avoidance phase. We transition to ACK-clocking rather than pacing the traffic with the system clock, but other forms of congestion avoidance would be possible.

III. Implementation

Paced chirping is implemented as two parts in Linux kernel version 4.13. [18] describes the implementation and the modifications we have made in more detail.

The first part is a modification to the internal pacing code introduced in version 4.13 that allows for inter-packet time gaps. Chirps are realized by controlling the gaps between subsequent packets. We have added two linked lists to the tcp sock structure that contains new and used entries with gap information for individual packets. This enables control of the earliest release time of each packet, similar to the timing wheels in Carousel [20], which could be used instead. Indeed, pressure is building to provide such a facility in hardware on the NIC [12].

The modification makes it possible to schedule chirps and guard intervals from a CC module. Figure 4 shows a schematic of the modification and the two lists we have added. Each time the kernel sends a packet it check if a new entry is available, and if so it applies the gap after the packet had been sent. It also writes a timestamp and the next sequence number to the entry before putting it in the linked list for used entries for examination by the CC module. The timestamps are used by the CC module to get the actual inter-send time of the packets. Small differences between requested and actual packet gaps are acceptable, but it is important that the CC module is made aware of the real gaps.

The second part is the addition of paced chirping logic to the DCTCP congestion control module that comes with Linux 4.13. DCTCPs congestion avoidance behaviour is unaltered. The code is available online[3].

In addition to the two main parts we have added three toggles as interim measures to disable delayed acks, hide ECN marks from the kernel stack, and disable kernel calculation of pacing rate. The first toggle is necessary as the kernel heuristic starts using delayed acks before paced chirping transitions to congestion avoidance. The second toggle prevents the kernel from entering congestion window reduction state upon receiving a single CE mark, but importantly the CE mark is still delivered to the congestion control module. The third toggle is necessary in the transition from paced chirping to congestion avoidance and ack-clocking.

IV. Evaluation

We have evaluated paced chirping in a physical testbed of 5 machines configured in a dumbbell topology to allow repeatable experimentation but with real equipment. It has two servers which act as traffic generators, an AQM which applies network characteristics, and two clients acting as sinks. The clients are connected to the AQM machine through a switch to allow them to share a bottleneck. All the machines are connected through an additional network to prevent control traffic from interfering with experimental traffic.

The AQM is configured using the tc command. Delay is introduced by netem, and rate limiting is applied by the hierarchical token bucket (HTB) qdisc with burst size limited to 1B.

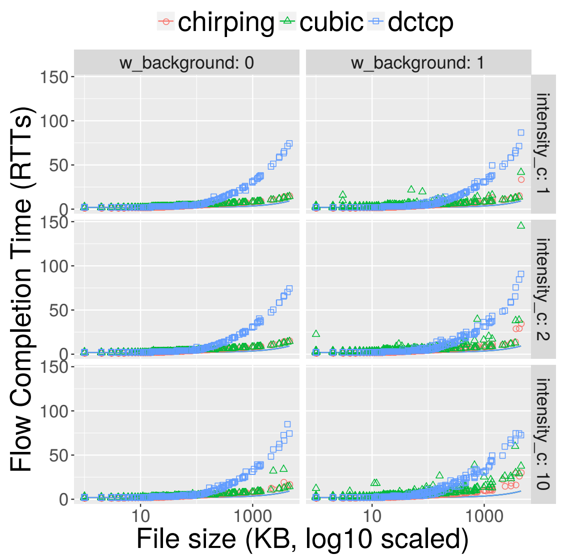
We will use paced chirping as a term for DCTCP with paced chirping instead of slow start. If nothing else is specified the initial gain and geometry are both set to 2. If nothing else is specified the ECN marking threshold is set to 1ms.

A. Flow Completion Time

The purpose of this experiment is to compare paced chirping with DCTCP and Cubic using more realistic web traffic. Each experiment has 1000 flows with Pareto distributed flow sizes mimicking recent real-world measurements since the introduction of HTTP/2 [17]. Alpha and mean are set to 0.5 and 900 Bytes respectively, and the sizes are limited to the range [1KB, 5MB]. The inter-arrival times of the flows are exponentially distributed with various mean values configured, called intensity. The network is configured with a 15 ms RTT and a 100 Mb/s capacity. DCTCP and paced chirping are run with a 1 ms marking threshold, while Cubic

[2]There also has to be enough pressure to trigger the scheduler of any shared links to open up the capacity, but we have not tested over such links yet.

[3]https://github.com/JoakimMisund/PacedChirping



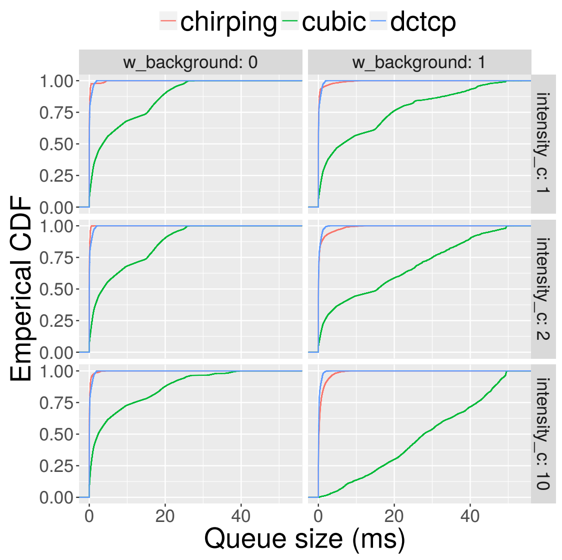
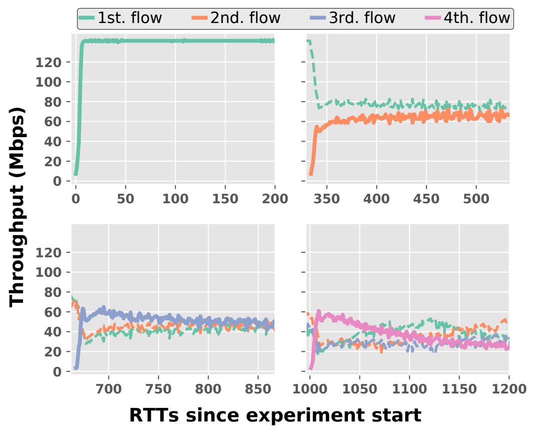


Fig. 5. Flow Completion Time and queueing delay for Cubic, DCTCP and Paced Chirping. DCTCP and Cubic makes opposite trade-offs between latency and FCT, while Paced Chirping achieves both low latency and good FCT. ECN-marking threshold 1 ms or 1 BDP tail-drop; RTT 50 ms; capacity 100 Mb/s.

is run with a tail-drop queue of 1 BDP. We ran Cubic with and without hystart [10] but, to avoid crowding the plots, we only show the results with hystart, because both FCT and queueing delay were slightly worse without it. We vary the intensity to test performance under various loads. The experiment is run with and without a greedy background flow.

Figure 5 shows the flow completion time and empirical CDF of the queueing delay of the 1000 flows under different conditions. Cubic achieves good FCT, but only at the expense of a significant queue. DCTCP does not handle the low marking threshold well and exits slow start early which reduces the FCT for longer flows. On the other hand the queuing delay is excellent. Flows that use paced chirping finish as fast as Cubic but with queueing delay nearly as low as DCTCP; far lower than Cubic. Paced chirping does not make the trade-offs that Cubic and DCTCP have to make.

B. *Time Series as 4 Flows Arrive*



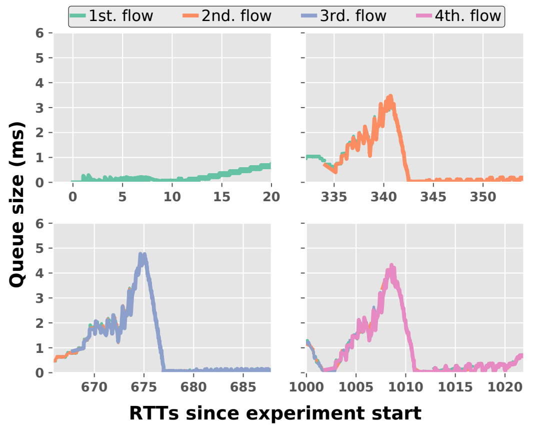


Fig. 6. Throughput and queue delay of 4 DCTCP flows starting one after another with paced chirping in an Internet environment. Each flow converges to steady state very fast with fairly low additional queueing delay. ECN-marking threshold 1 ms; RTT 15 ms; capacity 150 Mb/s.

The purpose of this experiment is to show visually how representative runs of paced chirping behave with and without background traffic and with varying number of flows.

Figure 6 shows 4 flows started after one another with enough time for the existing flows to converge before the next joins. This is the same experiment as used for motivation in subsection I-A except for the increased capacity.

The first flow accelerates right to the capacity without creating a noticeable queue except for the first two bursts of 5 packets. The flow successfully pulses the queue to get information about the capacity.

The other flows start in a fully utilized network and have to push the existing flows back more to claim a share of the bottleneck faster which builds slightly more queue than DCTCP does. It is possible to adapt the gain and geometry to make the flows less aggressive [18].

C. *Time series as Mixed Size Flows Arrive*

The purpose of this experiment is to visualize how paced chirping behaves in a run that is more representative of real user traffic, but still simple enough to see what is going on.

We add UDP traffic roughly equal to 20% of the total capacity. The inter-send time of the UDP-packets is exponentially distributed with an average corresponding to 20% of the total capacity. Then we add twelve 1 MB flows and two long lived flows all starting staggered. The network has a 20 ms RTT and 120 Mbps capacity.

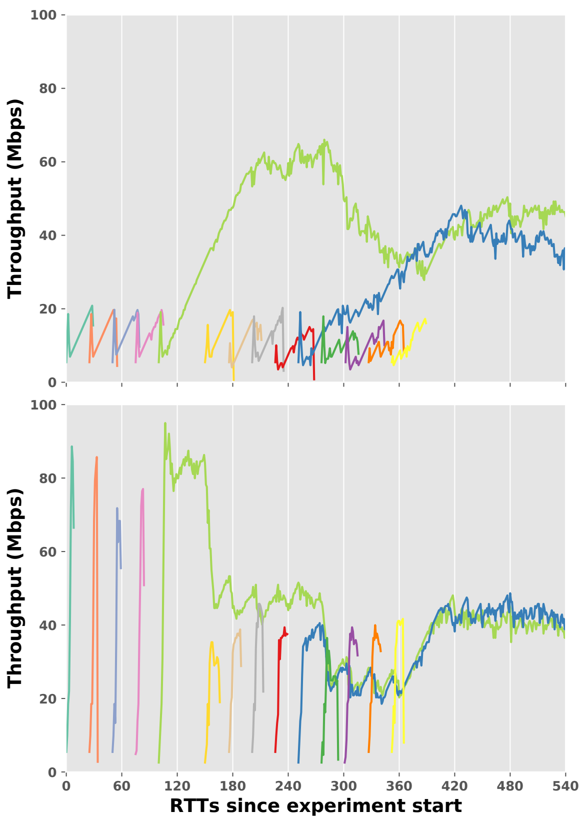


Fig. 7. Throughput of various flows including light UDP traffic (not shown). With paced chirping (bottom plot) the smaller flows complete faster and the long-lived flows accelerate faster relative to regular DCTCP using slow-start (top plot). ECN-marking threshold 1 ms; RTT 20 ms; capacity 120 Mb/s.

Figure 7 shows that paced chirping is able to accelerate faster than DCTCP which improves flow completion time and throughput when entering congestion avoidance.

V. Where Paced Chirping Fits

Paced Chirping is not only intended to be used during the initial start-up of a flow. It is as applicable during a restart after an idle, although we have not implemented that yet.

So-called scalable congestion controls such as DCTCP also offer much greater potential to exploit paced chirping at any point when the ECN signal disappears during congestion avoidance phase. By definition a scalable congestion control ensures that the frequency of congestion signal remains constant (at about 2 marks per RTT) as flow-rate scales. Thus the sender can detect when the signal has disappeared within a couple of RTTs. This most likely indicates that more capacity has become available, perhaps due to a flow departing or capacity increasing (e.g. radio links).

In contrast, with an ‘unscaleable’ congestion control it would be infeasible to trigger paced chirping after absence of any congestion signal had gone on longer than normal, because very long absences are already normal and they are getting longer as flow-rates scale. For instance Cubic sees a loss every 500 RTTs at 800 Mb/s over 20ms RTT.

Thus, with scalable congestion control, it is not necessary to chirp all the time during stable periods of congestion avoidance, as TCP RAPID [14] does. This avoids raising the noise floor during times when established flows have a frequent closed-loop ECN signal anyway.

VI. Conclusions

Paced chirping is a new and promising algorithm that seeks to replace slow start. Slow start is an open-loop algorithm with heuristics not suitable for low queueing delay environments. Paced chirping decouples the amount of data sent from the information it gets. Since paced chirping is purely delay-based it should be applicable to any environment, not just ECN-based ones.

A. *Future Work*

We divide future work into research and engineering.

a) Future research::

- Improving the noise filtering and precision of chirps, especially over variable-rate links, e.g. DOCSIS, GPON, LTE, WiFi, etc.
- Termination condition - when to stop pushing in
- Exploit ECN signals if available
- Best strategy during the first round
- Evaluate over wider range of conditions and iterate.

b) Protocol engineering::

- Handling delayed ACKs and ACK thinning, including the possibility of putting packet arrival time in stretch ACKs (an idea used in earlier versions of QUIC [11])
- Handling loss and reordering
- Interaction with TCP Fast Open.

References

[1] B. D. Adelstein, T. G. Lee, and S. R. Ellis. Head Tracking Latency in Virtual Environments: Psychophysics and a Model. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 47(20):2083–2087, 2003.

[2] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). *Proc. ACM SIGCOMM'10, Computer Communication Review*, 40(4):63–74, Oct. 2010.

[3] M. Alizadeh, A. Javanmard, and B. Prabhakar. Analysis of DCTCP: Stability, Convergence, and Fairness. In *Proc. ACM SIGMETRICS'11*, 2011.

[4] D. Black. Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation. Request for Comments RFC8311, RFC Editor, Jan. 2018.

[5] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I.-J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl. Reducing Internet Latency: A Survey of Techniques and their Merits. *IEEE Communications Surveys & Tutorials*, 18(3):2149–2196, Q3 2016. (publication mistakenly delayed since Dec 2014).

[6] B. Briscoe (Ed.), K. De Schepper, and M. Bagnulo. Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture. Internet Draft draft-ietf-tsvwg-l4s-arch-02, Internet Engineering Task Force, Mar. 2018. (Work in Progress).

[7] J. Carmack. Latency Mitigation Strategies. Blog post: https://www.twentymilliseconds.com/post/latency-mitigation-strategies/, 2013.

[8] K. De Schepper, O. Bondarenko, I.-J. Tsang, and B. Briscoe. 'Data Center to the Home': Ultra-Low Latency for All. Technical report, RITE Project, June 2015.

[9] K. De Schepper, B. Briscoe (Ed.), O. Bondarenko, and I.-J. Tsang. DualQ Coupled AQM for Low Latency, Low Loss and Scalable Throughput (L4S). Internet Draft draft-ietf-tsvwg-aqm-dualq-coupled-05, Internet Engineering Task Force, July 2018. (Work in Progress).

[10] S. Ha and I. Rhee. Hybrid Slow Start for High-Bandwidth and Long-Distance Networks. In *Proc. Int'l Wkshp on Protocols for Future, Large-scale & Diverse Network Transports (PFLDNeT'08)*, 2008.

[11] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-06, Internet Engineering Task Force, Sept. 2017. Work in Progress.

[12] V. Jacobson. Evolving from AFAP: Teaching NICs about time. https://www.netdevconf.org/0x12/session.html?evolving-from-afap-teaching-nics-about-time, July 2018.

[13] N. Khademi, D. Ros, and M. Welzl. The new AQM kids on the block: An experimental evaluation of CoDel and PIE. In *IEEE 17th Global Internet Symposium (GI 2014)*, Apr. 2014.

[14] V. Konda and J. Kaur. RAPID: Shrinking the Congestion-control Timescale. In *Proc. IEEE Conference on Computer Communications (Infocom'09)*. IEEE, Apr. 2009.

[15] M. Kühlewind and B. Briscoe. Chirping for Congestion Control - Implementation Feasibility. In *Proc. Int'l Wkshp on Protocols for Future, Large-scale & Diverse Network Transports (PFLDNeT'10)*, Nov. 2010.

[16] J. Kulik, R. Coulter, D. Rockwell, and C. Partridge. A simulation study of paced TCP. Technical Report NASA/CR-2000-209416, NAS 1.26:209416, E-12041, NASA, Jan. 2000.

[17] J. Manzoor, I. Drago, and R. Sadre. How HTTP/2 is changing Web traffic and how to detect it. In *Network Traffic Measurement and Analysis Conference (TMA), 2017*, pages 1–9, June 2017.

[18] J. Misund. Rapid acceleration in TCP Prague. Master thesis, University of Oslo, June 2018.

[19] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *Passive and Active Measurement Workshop (PAM'03)*, 2003.

[20] A. Saeed, N. Dukkipati, V. Valancius, C. Contavalli, A. Vahdat, et al. Carousel: Scalable traffic shaping at end hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 404–417. ACM, 2017.