Flow-start: Faster and Less Overshoot with Paced Chirping

Joakim Misund, Simula and Uni Oslo
<joakim.misund@gmail.com>
Bob Briscoe, Independent
<research@bobbriscoe.net>

IRTF ICCRG, Jul 2018
The Slow-Start Dilemma

- The more a flow accelerates
- The greater the overshoot of queuing delay
  - before the sender can notice one round trip later

- This is the received wisdom from slow-start in TCP
  - but it's a general dilemma for any capacity-seeking e2e transport:

- It's possible to sense when to stop earlier, using delay
  - but then it takes longer to converge (e.g. hybrid slow-start)

- Paced Chirping escapes this dilemma
Flow completion time and queue spike plots

- Intensity is mean flow inter-arrival time (Exp. distributed) [ms]
Applicability

• Solely delay-based, to be applicable to the Internet
  • cannot rely on special logic (AQM, ECN, etc.) at every possible bottleneck

• Most interesting where Q delay already ultra-low
  (’cos adds max 2-3ms\(^1\) pulses to queuing delay already present):
  • low latency with congestion control like DCTCP
    isolated from Classic TCP, e.g.
      – Data Centre\(^2\)
      – Internet with L4S DualQ Coupled AQM @bottleneck

• These are the environments we're interested in
  • but from limited testing it seems applicable to the general Internet too
  • try it for your environment – open sourced

\(1\) Over a 20ms base RTT path at 120Mb/s
\(2\) Can use shallow buffers without loss

AQM: Active Queue Management
ECN: Explicit Congestion Notification
DCTCP: Data Centre TCP
L4S: Low Latency Low Loss
Scalable throughput
Caveats

- This is research, not production-ready
- We've focused on proving the concept
- Many open issues for investigation / solution
  - Listed at end, but main ones are:
    1) Delayed ACKs & ACK thinning
    2) Bursty MACs and schedulers
Approach (1/3)

- Packet chirps
  - continually pulse queue by a few packets, then relax

- Samples available (and max) capacity
  - available: rate where ACKs spread from sent pattern (see next slide) (after filtering noise within chirp)
  - max: ACK rate of last 2 packets

- Maximizes ratio of capacity-information-rate to harm (queue delay)
  - run each (per chirp) measurement through EWMA
Using chirps to measure available capacity

- This example measures constant available capacity
- Code to interpret chirps filters noise to measure varying available capacity
Approach (2/3): paced chirps

- Avg rate of each chirp depends on EWMA of available capacity measured in previous rounds
- Noisy, but increasingly frequent measurements
- Queue delay solely depends on chirp geometry
- Notice, chirp length reduces
  - as available capacity measured in last round increases
Approach (3/3): adaptive gain

- Growth in #chirps per RTT depends on a gain variable
  - the more stable the measurements, the more gain increases (squeeze guard interval)
- Push-in a little harder than available capacity grows:
  - other flows yield
  - goal: activity-triggered link scheduler expands per-user capacity
- Still, queue delay solely depends on chirp geometry, not gain
- When to shift from paced chirps to ACK clocking?
  - when chirps fill the round trip
  - or …? (to be determined, perhaps using ECN for extra precision?)
A Whole Short Flow

- Inter-receive gap measured at sender using ACKS
Outcome

- Fast convergence
- Low queueing delay
Discussion

- **Slow Start vs. Pacing** – 2 extremes:
  - sending 2 pkts per ACK is a great way to build a queue
  - pacing at constant rate, increasing each round trip
    - if slightly below capacity: a great way to get no info about capacity at all
    - if slightly above capacity: a great way to build a huge queue

- **Line-rate Burst vs. Chirp**
  - chirp measures available capacity (and max)
  - ACK-rate from a burst only measures max capacity
  - Intuition
    - chirp fits increasing amount between existing pkts, and measures its ACK-rate when a queue started to build
    - burst squeezes as much as possible between existing packets, and its ACK-rate measures how fast the resulting queue drained
Where Paced Chirping Fits (1/2)

- No need to chirp during stable periods of congestion avoidance
  - each chirp is a signal for the sender, but noise for other flows
    - chirping elephants just confuse mice
  - using ACK clock reduces timer burden on servers
    (large majority of packets are sent in steady state)

- So, goal of paced chirping: “Do one thing and do it well”
  - reach smooth transition to closed loop (ACK-clocking, ECN, etc.)
Where Paced Chirping Fits (2/2)

- A building block, to replace all the instances of open loop capacity seeking:
  - flow start
  - re-start after idle
  - when congestion avoidance goes open-loop, e.g. another flow ends, radio capacity increases

- Future work: with Scalable CC, e.g. TCP Prague, DCTCP, etc.
  - after **2 round trips** without ECN-marks
  - start paced chirps to rapidly find new operating point

- Infeasible with unscaleable CC, e.g. **500-1000 round trips** for Cubic @800Mb/s
Much Further Work Needed

• Research
  • Termination condition – when to stop pushing in
  • Improving noise filtering & precision of chirps
    – esp. for bursty MACs: LTE/5G, DOCSIS, GPON
  • Exploiting ECN if available
  • Initial avg. gap for a wide range of possible networks
  • Evaluation over much wider range of conditions & iterate design
    – much lower/higher BDP, hi as well as lo stat. mux. bottlenecks, etc.

• Engineering
  • Delayed ACKs & ACK thinning
  • Handling loss, reordering during slow start
  • TFO when RTT estimate is stale in the first RTT
  • Apply the idea from QUIC where a stretch ACK lists arrival times
Design Contributions (1/2)

• Queue is independent of scale
  • depends solely on geometry of chirp, not on pkts per RTT

• Achieved by the guard interval betw. each chirp
  • if estimate of available capacity were perfect
    – back-to-back chirps should build a few pkts of Q, then relax it
  • guard interval allows for error in the estimation
    – the more consistent available capacity measurements are
    – the faster the guard-interval can shrink
    – and the faster the congestion window can grow
Design Contributions (2/2)

- Paced chirping is continually crafting the packets it sends
  - around its measurements in previous rounds
  - while allowing for the possibility of change and error
- It is simultaneously
  - evolving its estimate of available capacity
  - sending packets at a spread of rates around this
  - sampling at multiple points across each round trip time
  - pushing back possible competing traffic
  - tracking variability of its measurements
  - reducing its guard interval accordingly
  - and therefore increasing its window
- Paced chirping shrinks the measurement-response loop
  - it makes flow-start closed-loop
Contributions to Experimentation

- Open sourced
  - github.com/JoakimMisund/PacedChirping

- Implementation in Linux
  - based on internal pacing in 4.13 kernel (earliest with internal pacing)
  - added kernel support for list of inter-packet times
  - paced chirping kernel module

- Advice on how to suppress certain kernel assumptions
  - disabled kernel pacing rate calculation
  - made it possible to hide CE marks from the TCP stack

- Published initial evaluation
Summary

• TCP slow-start is mimicked in most transport protocols
  • an open loop phase characterized by arbitrary numbers

• Paced chirping
  • closes the open loop – frequent startup information
  • queue delay solely depends on geometry of each chirp, not pace of chirps
  • Aims to maximize ratio: (capacity-information-rate / harm)

• Initial research
  • much more testing and development to do
Flow-start: Faster and Less Overshoot with Paced Chirps

Q&A
and
Spare Slides
Delayed ACKs & ACK thinning

- Could put arrival times in ACKs (as in QUIC)
- Could introduce a 1-bit option for sender to request quickacks
- Failing either, we need rcvr to suppress delayed ACKs during SS
  - Linux rcvr quickacks during SS
  - but paced chirping confuses its heuristics
Measuring Available Capacity using Chirps

• Find inter-packet gap where path delay starts to persistently increase

1) Record each inter-packet path delay increase

\[ \Delta q_n = q_n - q_{n-1} \]

where \( q = ts_{rcv} - ts_{snd}; ts \) are timestamps; and \( n \) is the packet number

2) Ideally one-way delay: timestamp each packet:
   • when sent (not when scheduled to send)
   • when received
     − in practice use when ACK rec'd (round trip delay)

3) Filter out noise. Simple example filter:
   • only count an increasing trend of more than \( L \) packets
     to count as an increase, \( \Delta q > \frac{\max_{i=1}^n (\Delta q_i)}{F} \)
   • default: \( L=5, F=1.5 \)
Linux Pacing Framework

struct tcp_sock

- pacing_timer -> expire
- used_pacing_rate_list
- pacing_rate_list

1. Use rate if avail.

2. Put used rate

tcp_internal_pacing

tcp_transmit_skb

tcp_pacing_check

tcp_write_xmit
	1. tcp_pace_kick
	schedules

modifications in red
Linux kernel
Structure to set up per-packet rates

```
tcp_internal_pacing
1. Check if available rate
2. Use rate, put time and snd_nxt
3. Move entry to used-list
```
Escaping the Slow-start dilemma

- Probably broken version of BBR over v4.13 kernel
Higher intensity;
Paced Chirping needs a better termination condition