Paced Chirping: Rethinking TCP start-up

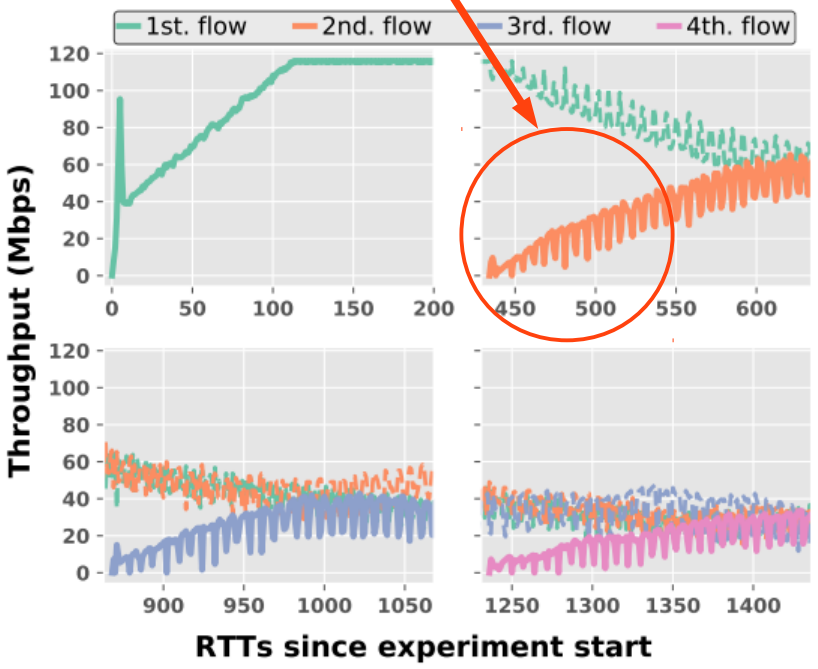
Joakim Misund (University of Oslo)
Bob Briscoe (Independent)

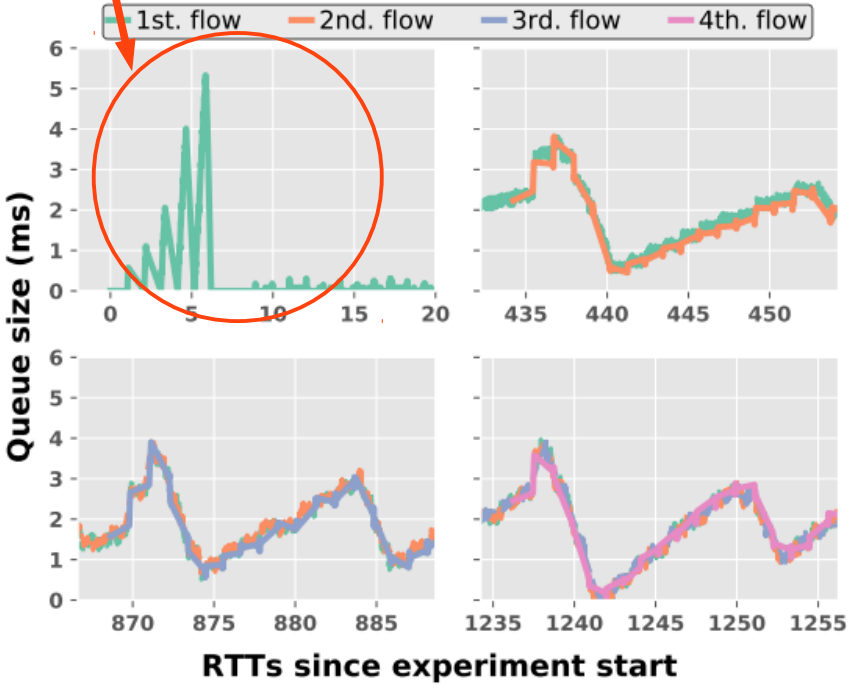
TCP Slow Start

- Roughly double the number of packets each round-trip time
  - Transitions to congestion avoidance on first loss/ECN
- Exponential increase + lag in reaction → Queue Overshoot
  - Increased delay and congestion events
- Trade-off between ramp up speed and queue impact
  - The potential blowback grows with increasing speed
- Overly sensitive?
  - No way to recover from bad luck
  - Undershoot in ultra low queuing delay scenarios (L4S, DCTCP)
- Optimal: Get up to speed fast while keeping maximum queueing delay low.

Undershoot in low Q scenarios

Spikes of queueing delay





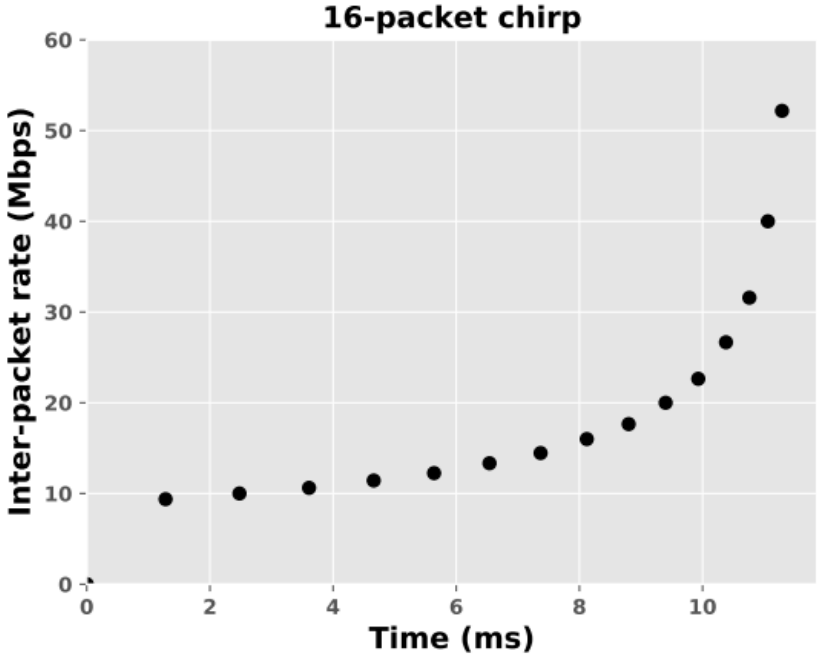
Imagine only 1ms queue. First flow would terminate prematurely in second round

Paced Chirping

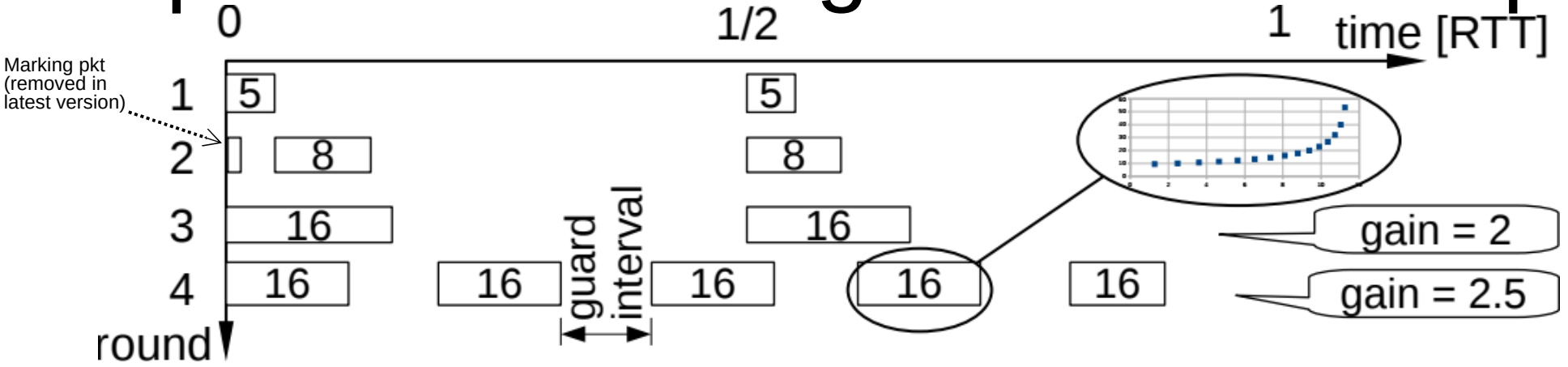
- Estimates available capacity as the flow accelerates
- Useful at flow-start and after idle-periods
  - And congestion avoidance in L4S (see later)
- Primarily delay-based
  - Widely applicable
- Paced Chirping is NOT just for L4S!
  - Should be applicable in any CC
  - Possibly better accuracy in L4S (ECN)
- Goal is to eventually become a closed-loop start-up algorithm

Chirp

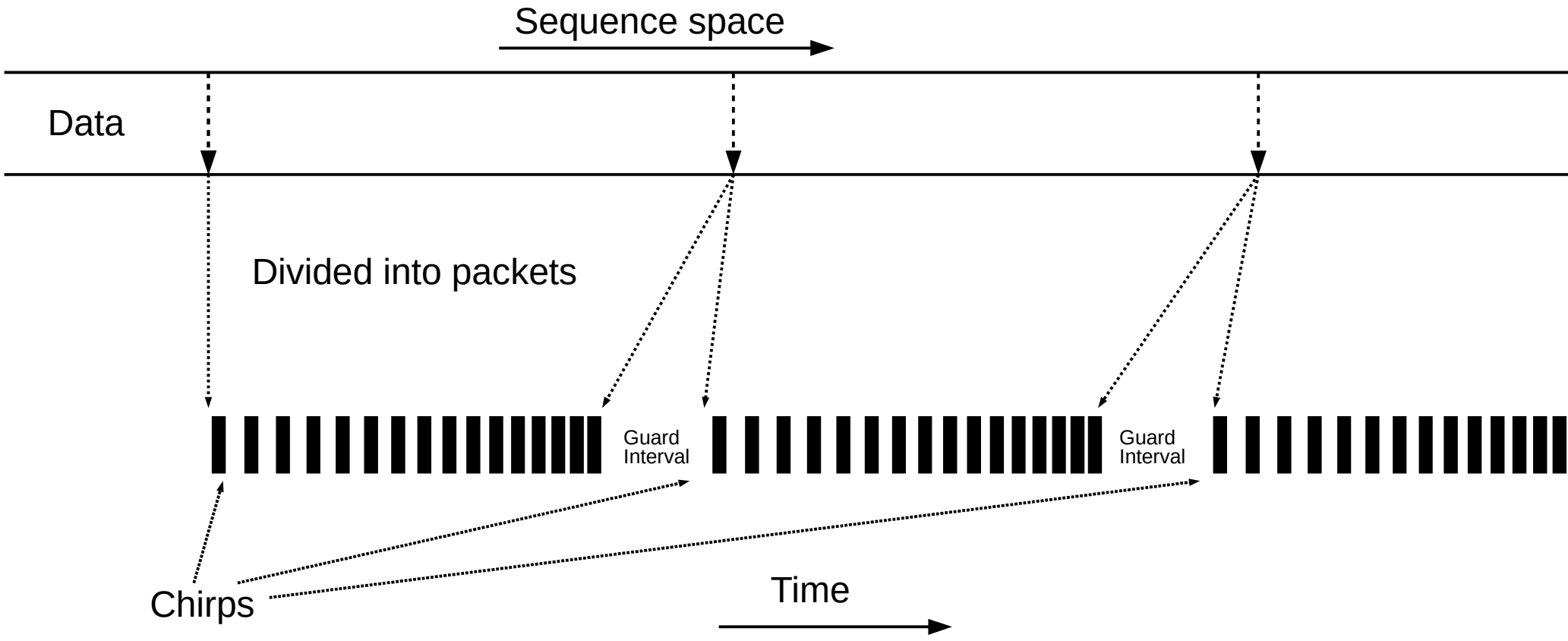
- Group of packets sent at an increasing rate
  - Realized by decreasing inter-packet gap
- Known average inter-packet gap
- Spread controlled by geometry (variable)
  - Determines max queuing delay
- Estimate available capacity using measured relative queueing delay of packets
  - Looking for increasing trends in queueing delay



Spread increasing number of chirps

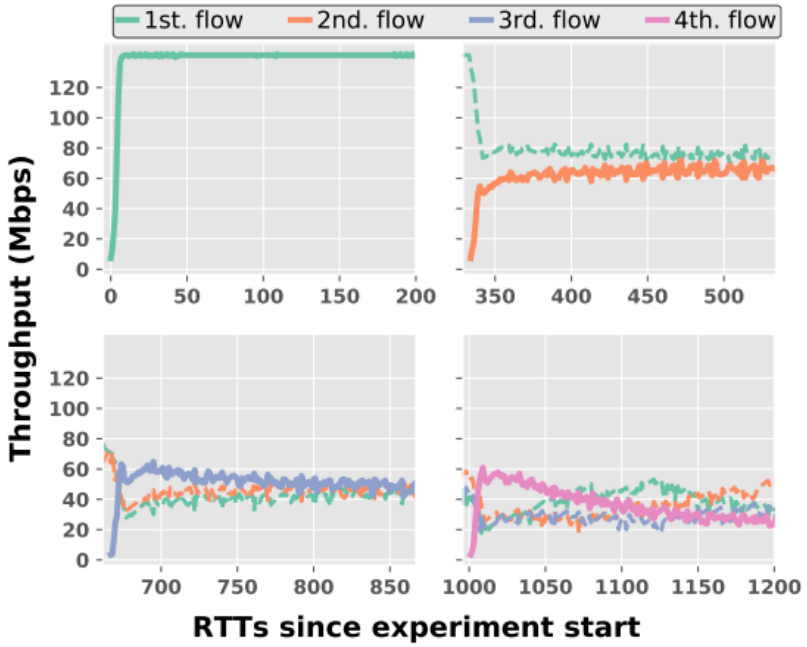


- M chirps in a round. M set to M * gain each round starting from round 4
  - Maximum queue delay depends on geometry, not number of chirps
- Guard interval allows queue to relax in-between chirps
  - Allows for error in estimate
- Avg. gap of chirps depend on previous estimates. Except first two
- Terminates once the ‘pipe is full’. Chirps fill round-trip-time
- Taking longer is not necessarily more careful, because the first tests become more stale by the time you get to the last test

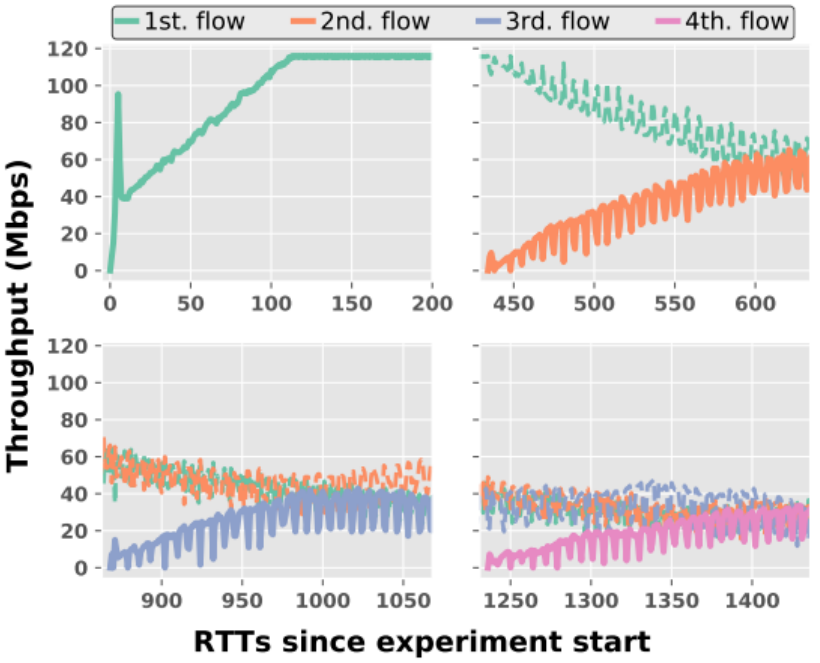

Sequence space
Data
Divided into packets
Guard Interval
Guard Interval
Chirps
Time

What we can get

With PC



Without PC



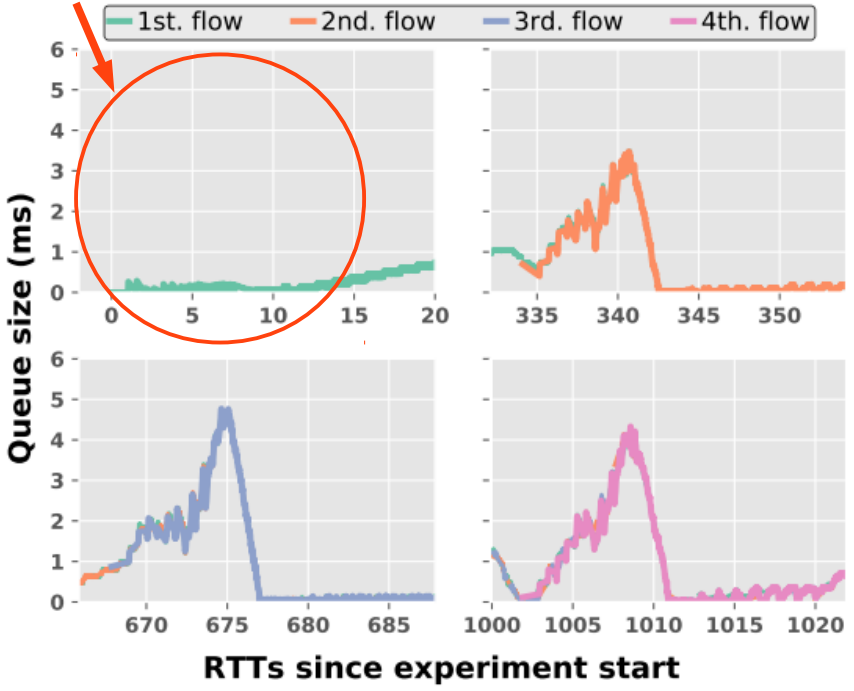
- Pushes existing flows back
  - With little extra queueing delay (next slide)

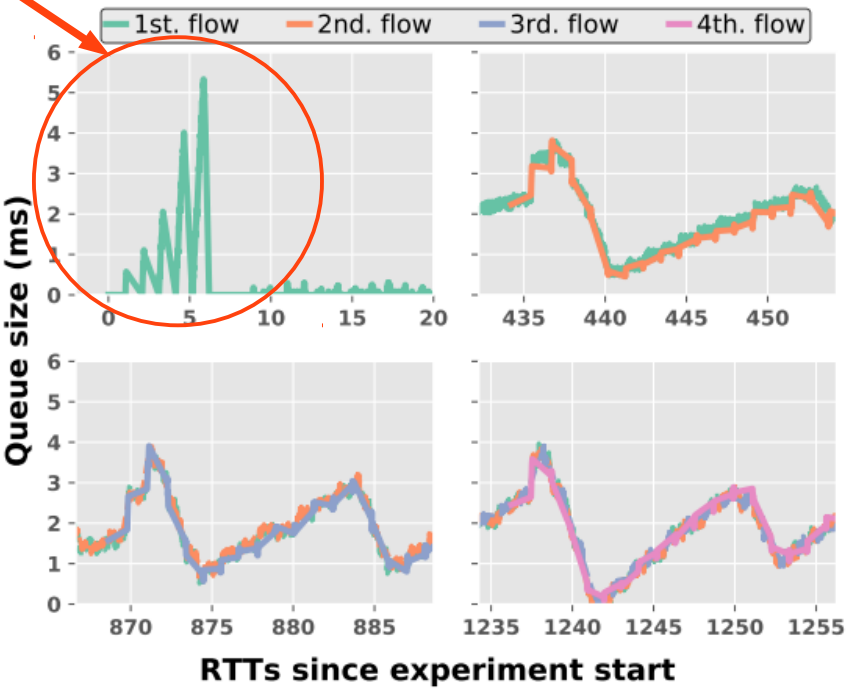
With PC

Without PC

Hardly any queue

Bursty behaviour





L4S

- Low Latency Low Loss Scalable throughput (L4S)
- Early and aggressive ECN marking in network (Low latency and Low Loss)
- End system reacts to aggregate of ECN marks, rather than presence → Small changes to cwnd (Scalable Throughput)

Congestion Avoidance

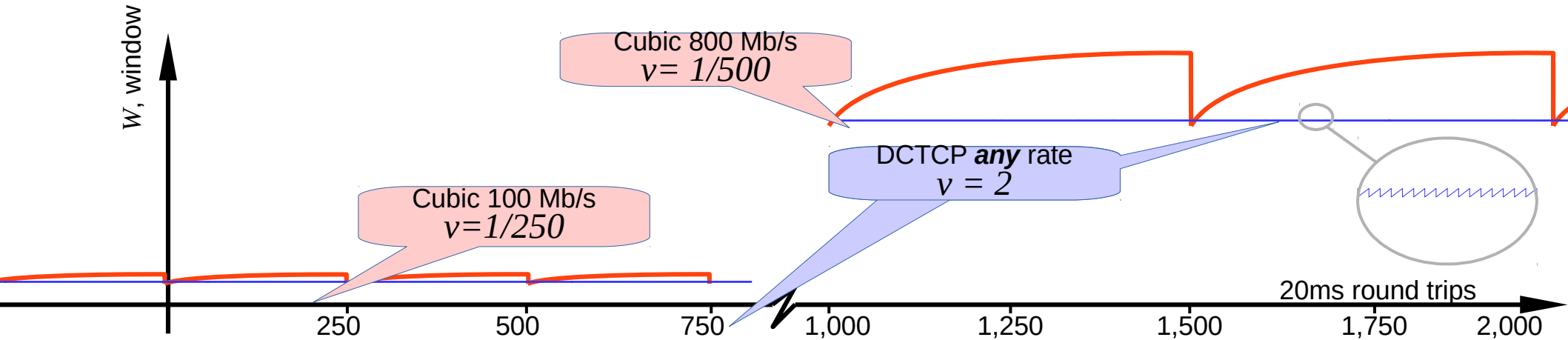
- Slow at acquiring available capacity. E.g when competing flow(s) leave
  - Careful probing (additive, cubic)
- Traditional TCP does not know if in-between congestion events or if more capacity has become available
  - Few congestion events per RTT needed to keep throughput high

BDP = Bandwidth-Delay Product

Lesson from design of high-BDP TCP protocols

- keep the number of round trips between drops small
- equivalently, keep the number of drops per round trip large

v : number of congestion signals per round trip



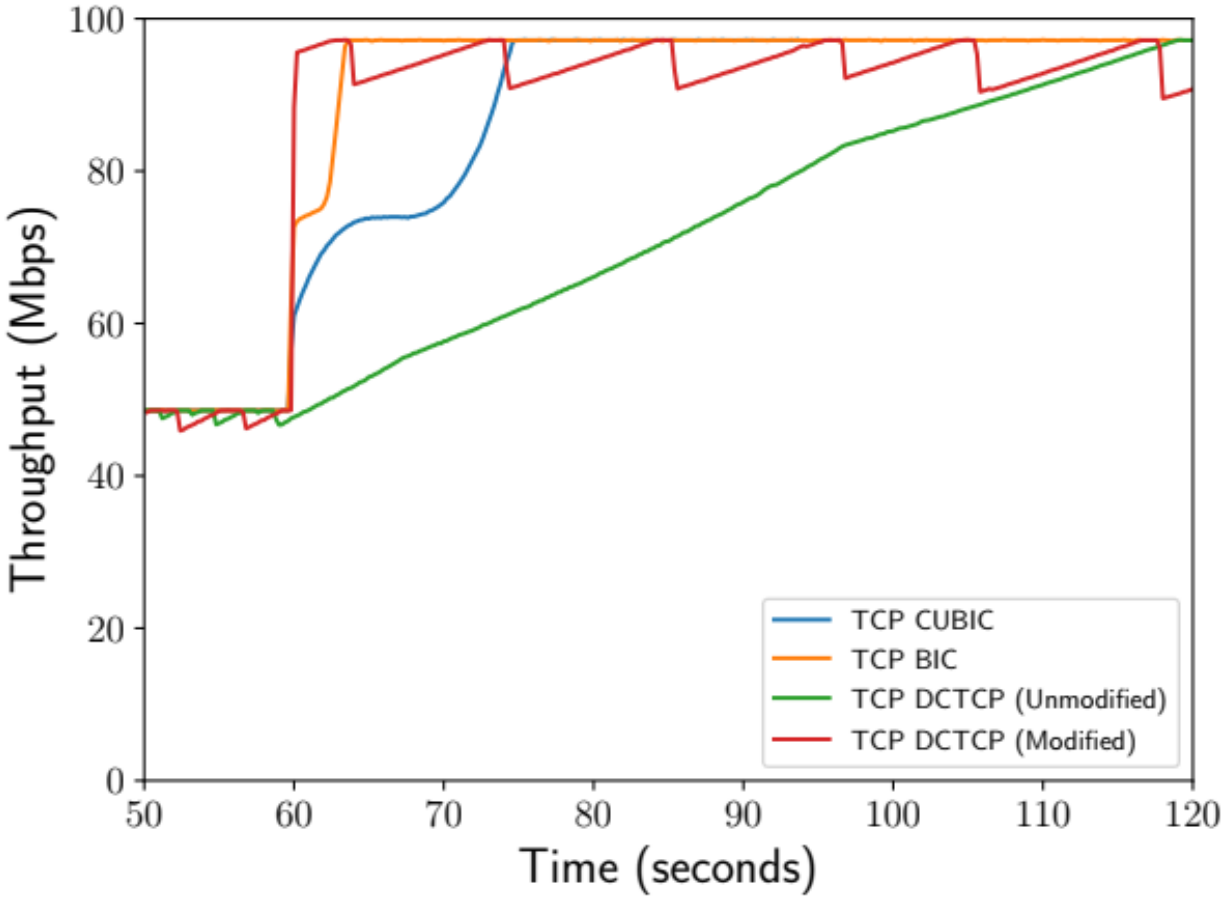
Taken from (with permission): https://www.bobbriscoe.net/presents/1707ietf/1707tcp-prague-scaling-tensions.pdf

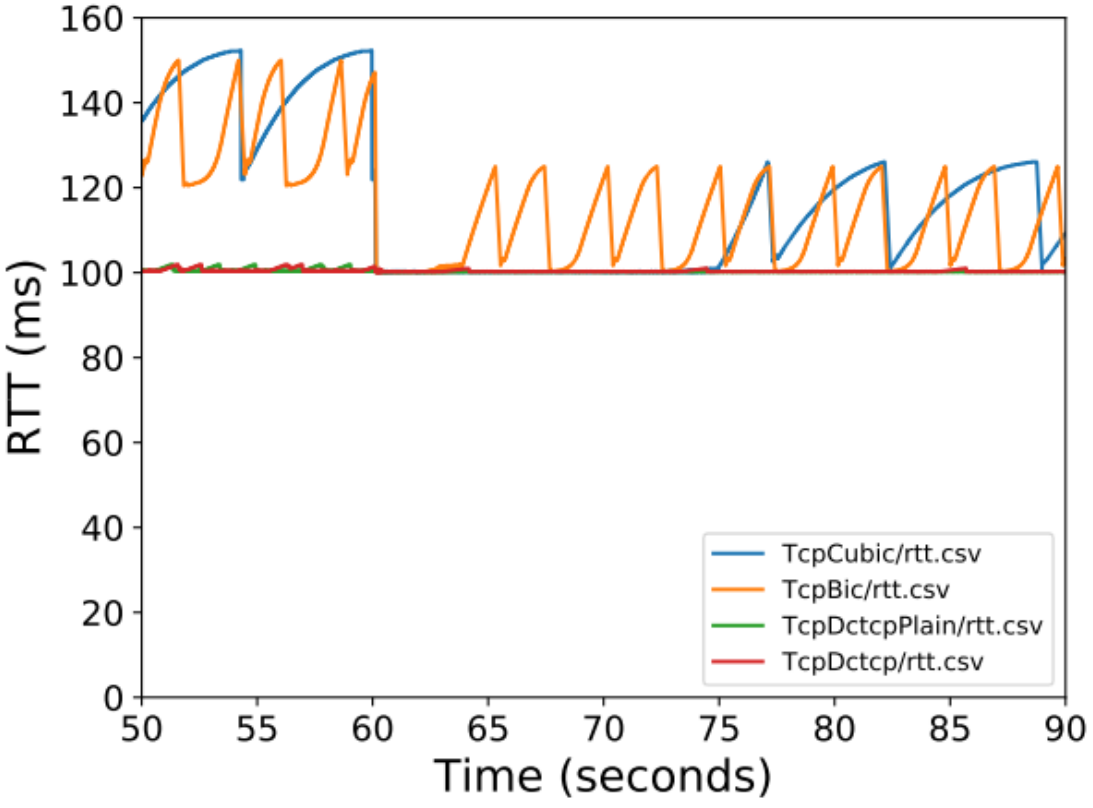
Congestion avoidance in L4S

- Detecting increase in available capacity made possible by frequent marks in L4S.
  - If normally 500 RTTs between marks, it takes ~1000 RTTs to notice their absence
  - If normally 2 marks per RTT, it takes ~1 RTT to notice

NS-3 simulation

- RTT: 100ms
- Capacity 50Mbps → 100Mbps
- Qlen:
  - Dctcp 1ms
  - (Cu)bic .5 BDP

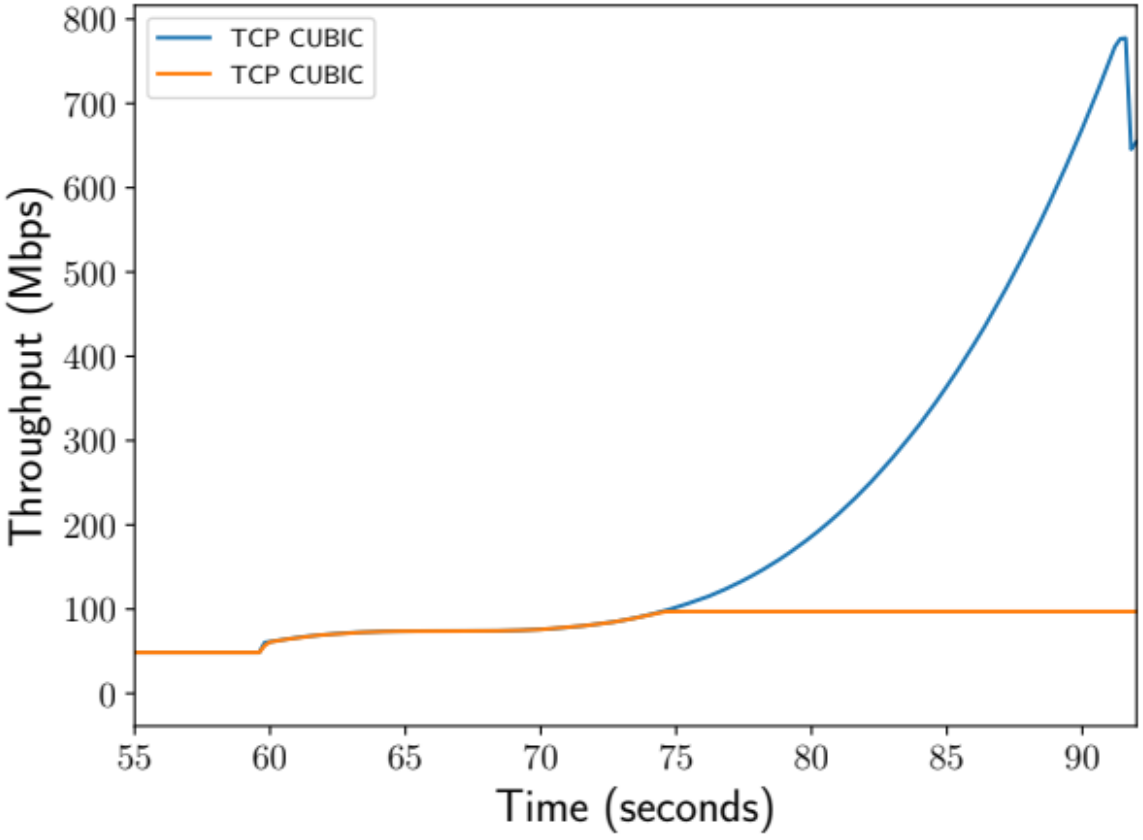



160
140
120
100
80
60
40
20
0
RTT (ms)
50
55
60
65
70
75
80
85
90
Time (seconds)
TcpCubic/rtt.csv
TcpBic/rtt.csv
TcpDctcpPlain/rtt.csv
TcpDctcp/rtt.csv

Is Cubic scaleable?

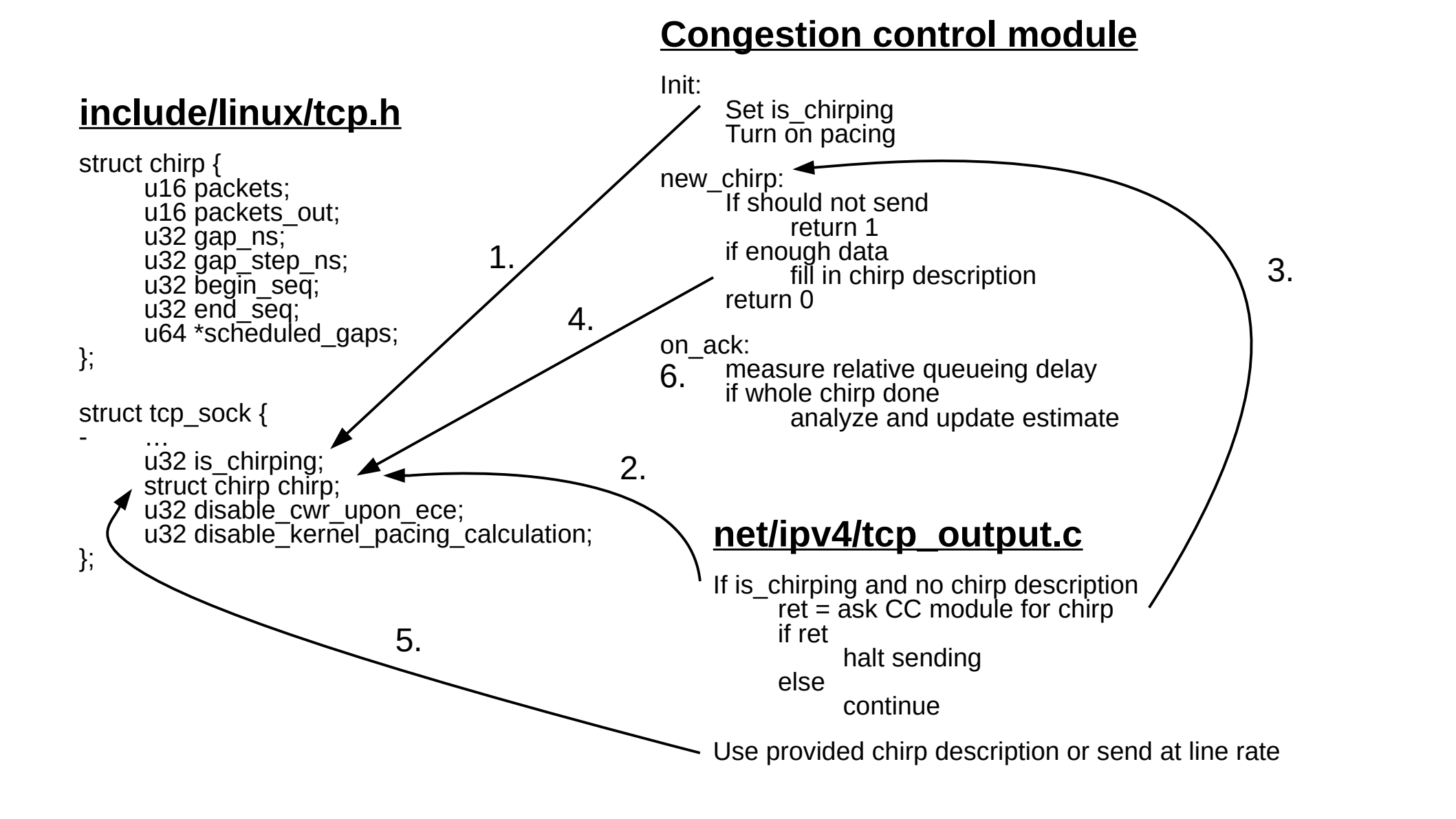
- RTT: 100ms
- Capacity changes

  - 50Mbps → 100Mbps
  - 50Mbps → 800Mbps
- Takes roughly twice as long



Implementation

- Code available at github.com/JoakimMisund/PacedChirping
- Kernel changes
  - New struct describing a chirp in struct tcp_sock
  - Flag for enabling chirping in struct tcp_sock
  - New CC ops callback for getting chirp
  - Logic for realising chirps in tcp_output.c
- Paced Chirping logic in DCTCP CC module
  - Seems difficult to make general. Each CC needs (possible shared) implementation.


Congestion control module
Init:
Set is_chirping
Turn on pacing
new_chirp:
If should not send
return 1
if enough data
fill in chirp description
return 0
on_ack:
measure relative queueing delay
if whole chirp done
analyze and update estimate
include/linux/tcp.h
struct chirp {
u16 packets;
u16 packets_out;
u32 gap_ns;
u32 gap_step_ns;
u32 begin_seq;
u32 end_seq;
u64 *scheduled_gaps;
};
struct tcp_sock {
- …
u32 is_chirping;
struct chirp chirp;
u32 disable_cwr_upon_ece;
u32 disable_kernel_pacing_calculation;
};
net/ipv4/tcp_output.c
If is_chirping and no chirp description
ret = ask CC module for chirp
if ret
halt sending
else
continue
Use provided chirp description or send at line rate
1.
2.
3.
4.
5.
6.

Using it

- 1. Install Kernel
- 2. Create CC module that implements the new cc_ops callback and
  - set is_chirping to 1.
  - enable pacing by setting sk_pacing_status
- 3. Provide the kernel with chirp description through implementation of new_chirp callback

Further work

- Variable-rate links e.g DOCSIS and WiFi
- More evaluation
- Exploit ECN
- Handle delayed ACKs and ACK thinning
- Handle loss and reordering

Questions?

FAQ

- Pacing precision is surely a challenge. How does Paced Chirping deal with this?
  - Paced Chirping is insensitive to the actual send-time of each packet as long as the gaps are decreasing.
- Line bursts are simpler to implement, why are chirps necessary?
  - Line bursts measure the maximum capacity, while chirps measure the available capacity.
- Why not add special acceleration signals to every L4S AQM?
  - Not incrementally deployable
    - even if a special node signals 'accelerate by x'
    - in the next round a non-special node could become the bottleneck