Reducing Latency for Linux Transport

Per Hurtig
Karlstad University

Andreas Petlund
Simula Research Laboratory

Dublin 06.10.2015

RITE – Reducing Internet Transport Latency

The EU-project RITE : Partners

Industry partners:

- British Telecommunications (UK)
- Alcatel-Lucent Bell (BE)
- Megapop (NO)

Academic partners:

- Simula Research Laboratory (NO)
- University of Oslo (NO)
- Karlstad University (SE)
- Institut Mines-Telecom (FR)
- The University Court of the University of Aberdeen (UK)

Internet Latency

Limitations of scope

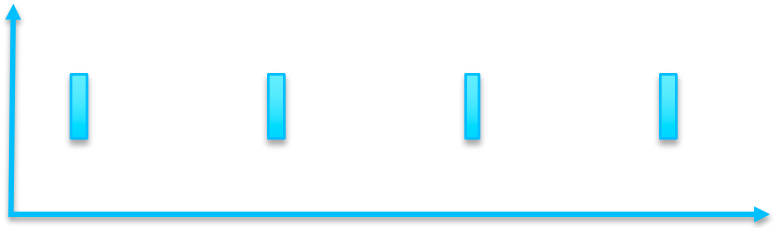
- The mechanisms we'll talk about is about
  - reliable, congestion-controlled transport
  - avoiding retransmissions
  - avoiding time wasted on getting up to speed
  - minimise queueing delay

Traffic patterns matter to latency

Short flows

Web traffic – most Internet traffic

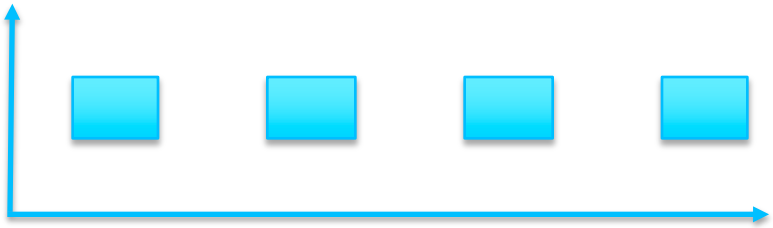
RTO Restart / TLP Restart



Thin streams

Interactive, real-time, sensors, games

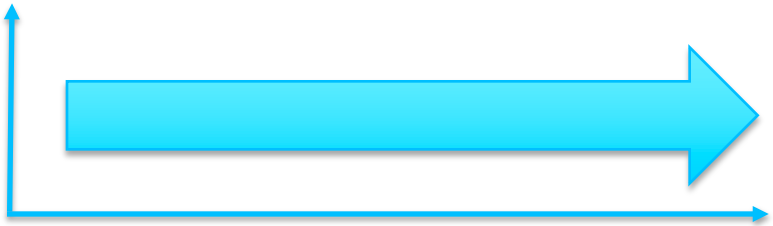
Redundant Data Bundling (RDB)



Bursty flows

HTTP segment streaming (Netflix)++

New CWV

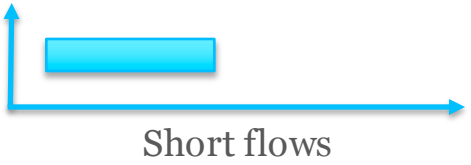


Greedy flows

Downloads

CAIA Delay Gradient (Linux edition)

TCP Tail Loss Recovery



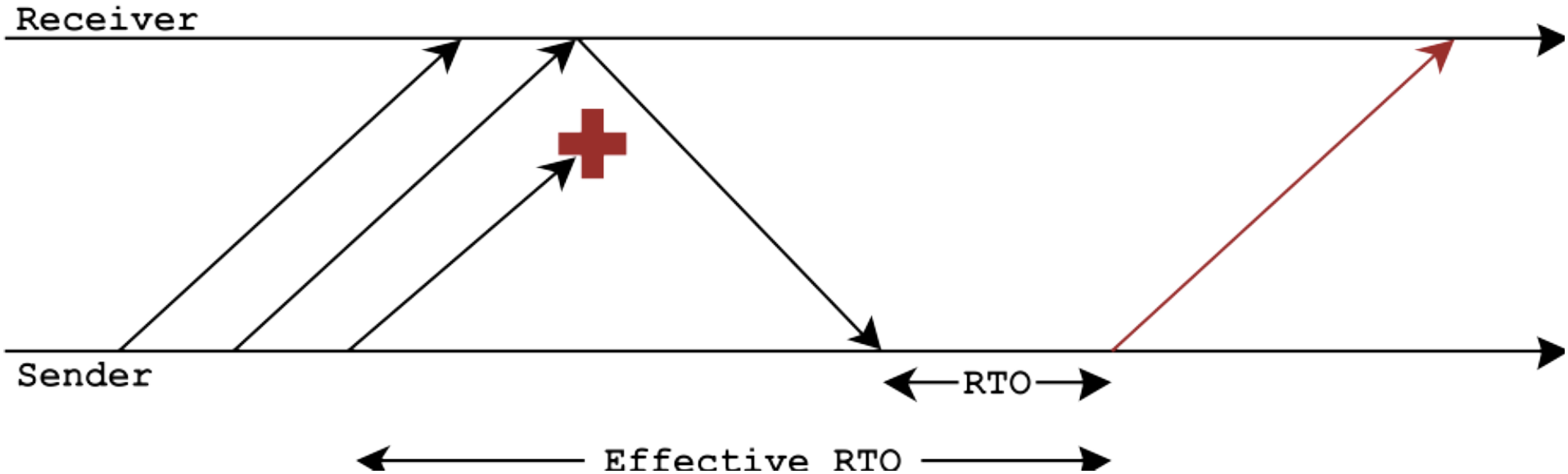
- The tail of transfers/bursts in TCP is critical for low latency
  - cannot use fast/early retransmit (FR/ER) for loss recovery
- For short and/or bursty flows this is really bad
  - tail constitutes large part of transfer
  - low latency is often important for applications sending this type of traffic

TCP Tail Loss Recovery

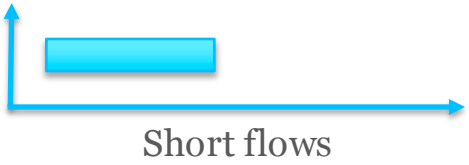
- A retransmission timeout (RTO) is used if FR/ER cannot be used
- RTO is a slow recovery mechanism based on the round-trip time (RTT) of the connection
- An RTO will cause larger congestion control impact than FR/ER

More Problems...

Short flows



RTO Restart (RTOR)



- An alternative way to restart TCP's RTO timer
  - removes the unnecessary offset
- RTOR is defined in "draft-ietf-tcpm-rtorestart-08"
  - approved by IETF for publication

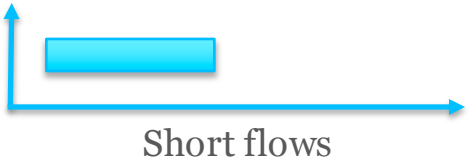
The Solution

When restarting the RTO, set:

```
RTO = RTO – T_earliest
```

where `T_earliest` is the transmission time of the earliest outstanding segment.

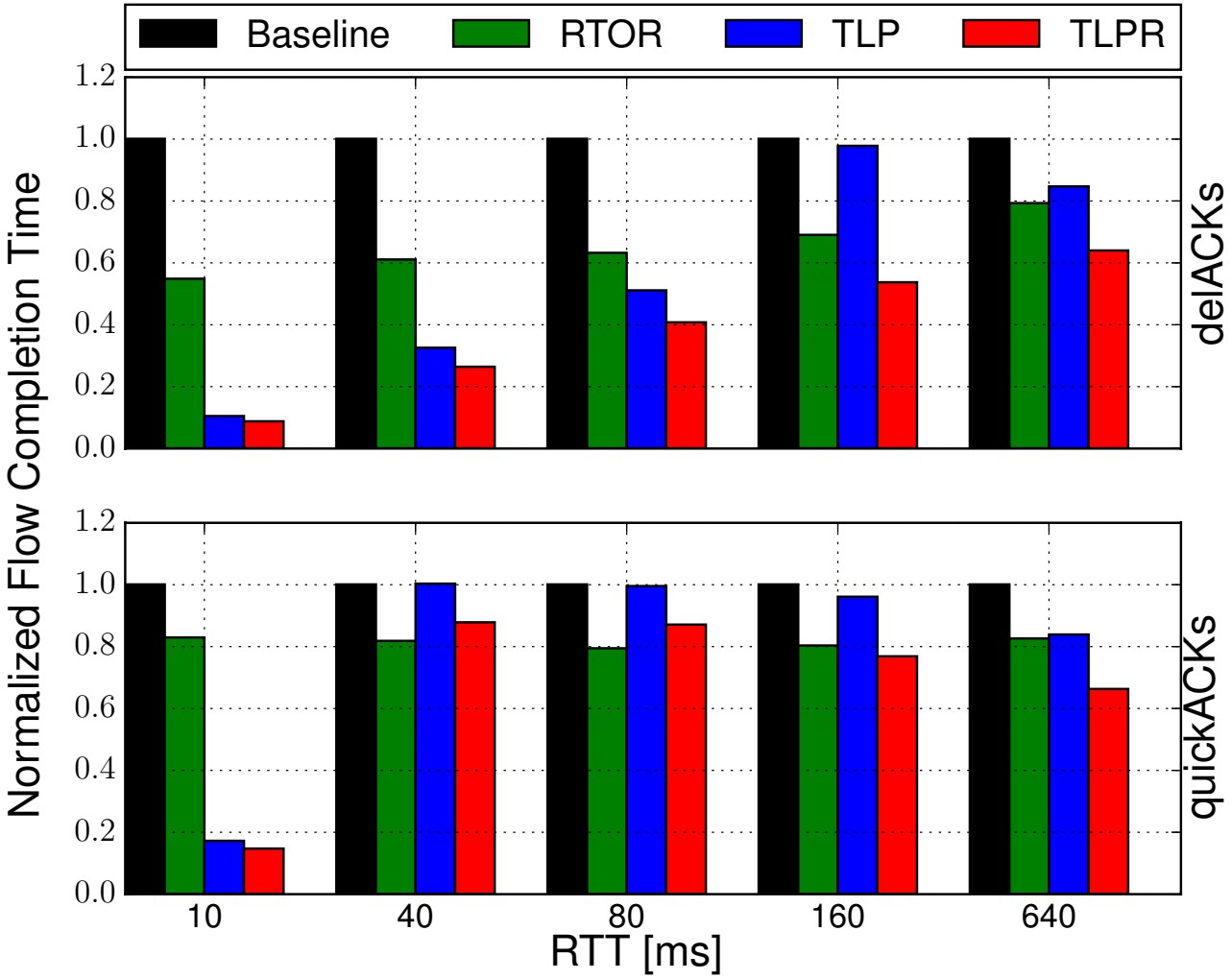
Tail Loss Probe Restart (TLPR)



- TLP is the “Linux” way of recovering from tail loss
- TLP tries to send new data/retransmit the latest transmitted segment on timeout
  - to trigger fast recovery instead of RTO
- The restart logic is the same as for the RTO
- TLP is defined in: “draft-dukkipati-tcpm-tcp-loss-probe-01”

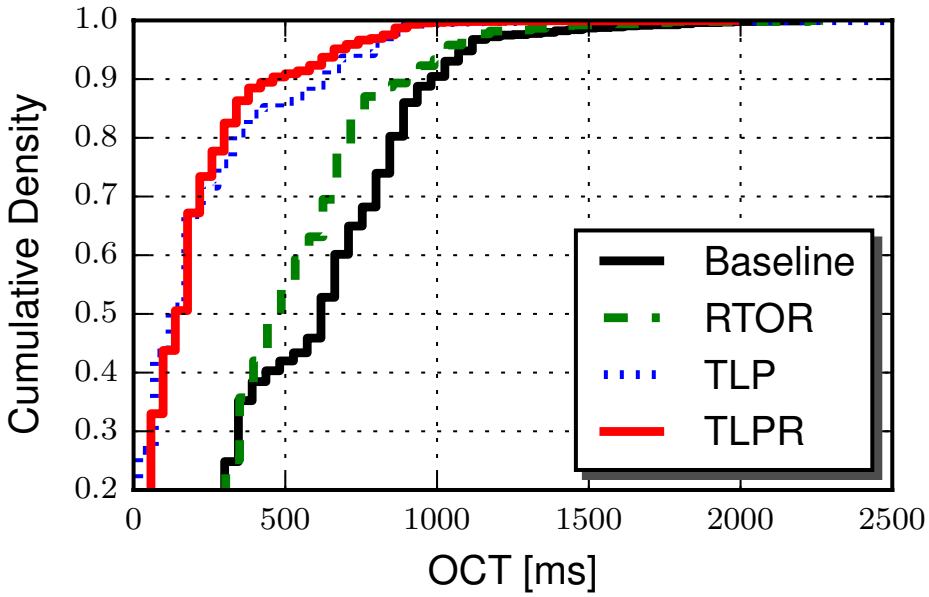
Losing the last segment

Short flows

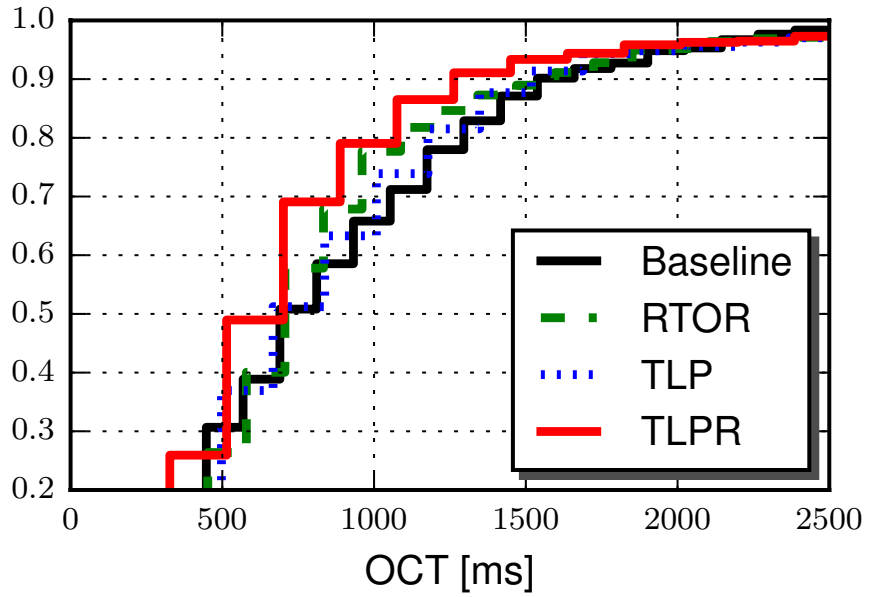


Web Page Downloads

Short flows



10ms RTT



160ms RTT

Changes to kernel (RTOR)

net/ipv4/tcp_input.c

```
void tcp_rearm_rto(struct sock *sk)
{
    […]
    else if (icsk->icsk_pending == ICSK_TIME_RETRANS &&
            sysctl_tcp_rto_restart &&
            tp->packets_out < sysctl_tcp_rto_restart &&
            (tp->packets_out + tcp_unsent_pkts(sk) <
             sysctl_tcp_rto_restart)) {
        struct sk_buff *skb = tcp_write_queue_head(sk);
        const u32 rto_time_stamp = tcp_skb_timestamp(skb);
        s32 delta = (s32)(tcp_time_stamp – rto_time_stamp);

        if (delta > 0 && rto > delta)
            rto -= delta;
    }
    inet_csk_reset_xmit_timer(sk, ICSK_TIME_RETRANS, rto,
                              TCP_RTO_MAX);
}
```

Changes to kernel (TLPR)

net/ipv4/tcp_output.c

```
bool tcp_schedule_loss_probe(struct sock *sk)
{
    […]
    if (tp->packets_out == 1)
        timeout = max_t(u32, timeout,
                        (rtt + (rtt >> 1) + TCP_DELACK_MAX));

        const u32 pto_time_stamp = tcp_skb_timestamp(skb);
        s32 delta = (s32)(tcp_time_stamp – rto_time_stamp);

        if (delta > 0 && timeout > delta)
            timeout -= delta;

    timeout = max_t(u32, timeout, msecs_to_jiffies(10));
    […]
    inet_csk_reset_xmit_timer(sk, ICSK_TIME_LOSS_PROBE, timeout,
                              TCP_RTO_MAX);
    […]
}
```

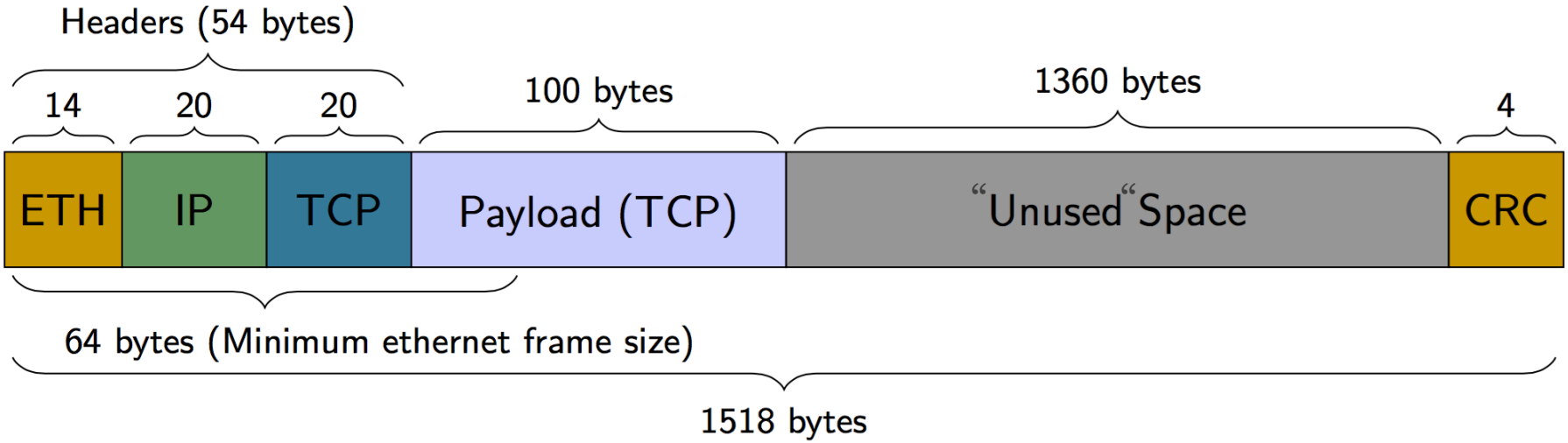
Redundant data bundling (RDB)

- Thin streams:
  - small packets
  - (relatively) high inter-transmission times between packets.
  - latency sensitive (traffic patterns arise due to timing/events/interaction)
- No backpressure →Not able to trigger fast retransmit

Application	Payload size			Packet inter-arrival time (ms)			
	avg	min	max	avg	med	min	max
Windows Remote Desktop	111	8	1417	318	159	1	12254
VNC (from client)	8	1	106	34	8	0	5451
VNC (from server)	827	2	1448	38	0	0	3557
Skype (2 users)	236	14	1267	34	40	0	1671
SSH text session	48	16	752	323	159	0	76610
Anarchy Online	98	8	1333	632	449	7	17032
World of Warcraft	26	6	1228	314	133	0	14855
Age of Conan	80	5	1460	86	57	0	1375

RDB – main principle

Thin streams

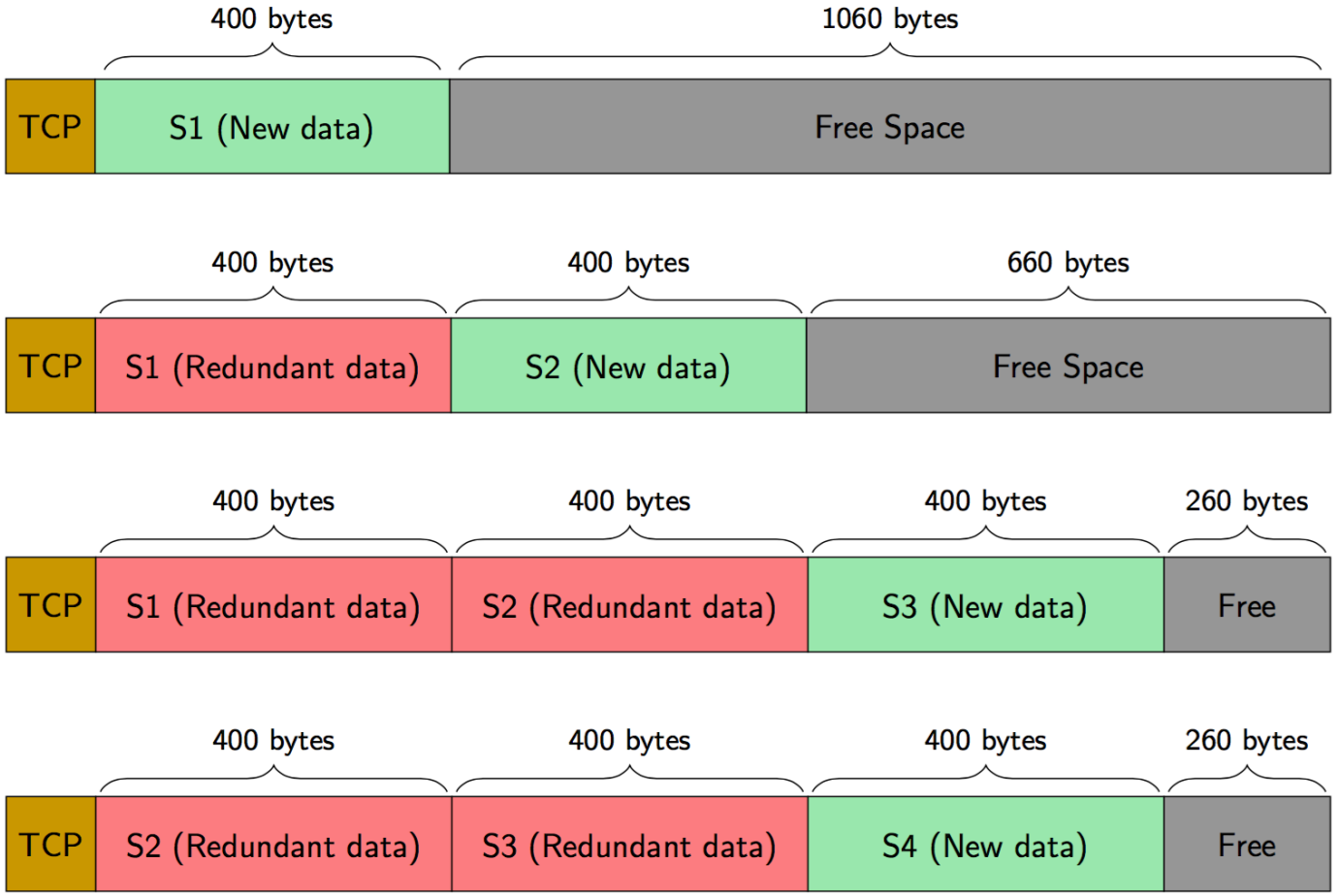


typical ethernet frame

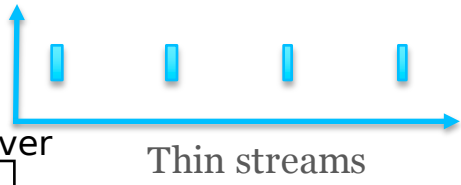
- send redundant data, but never send more packets.
- sender-side only mechanism.
- reduces retransmission latency and head-of-line blocking delay

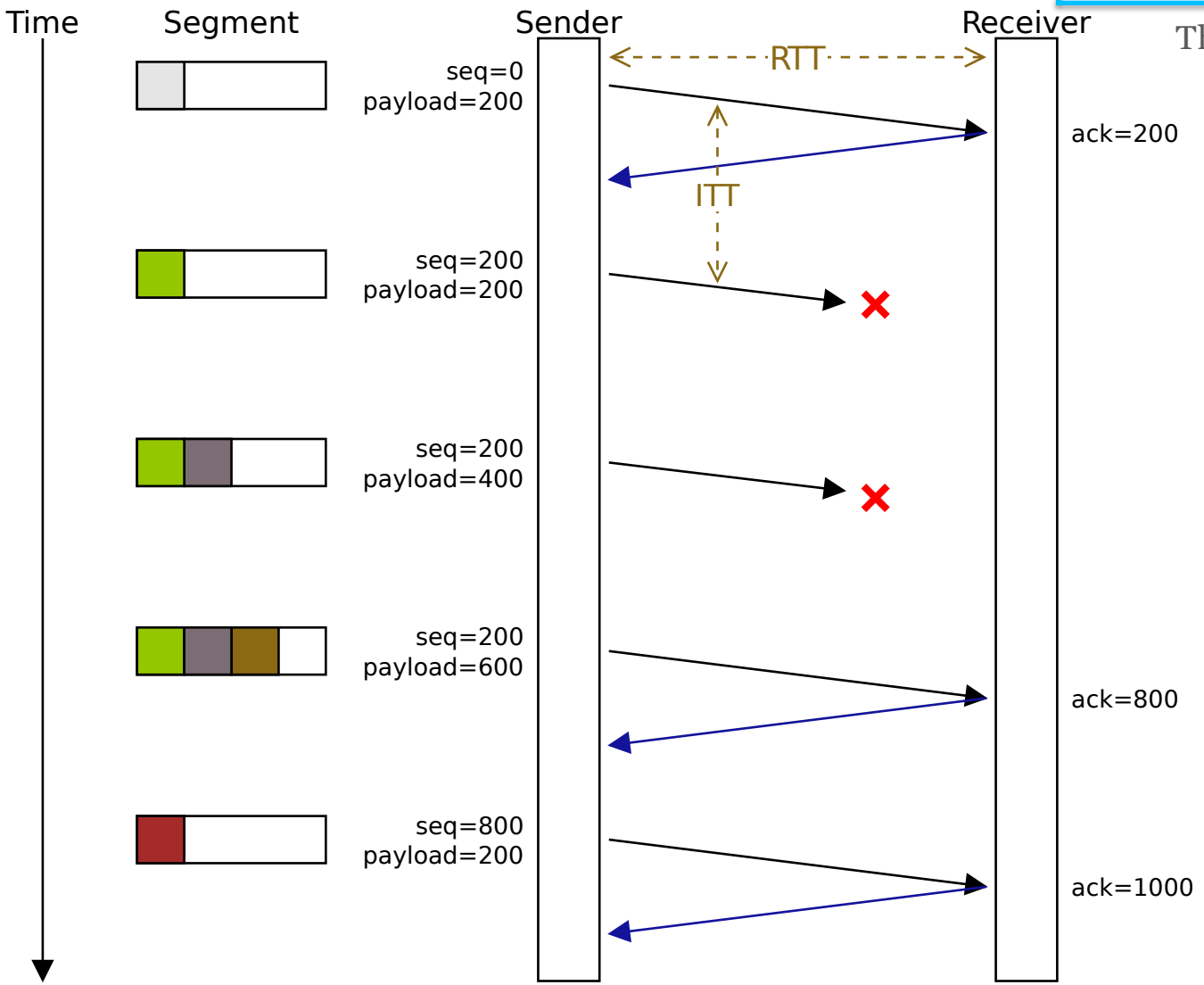
RDB – main principle

Example with four separate data segments showing how RDB organizes the data in each packet.

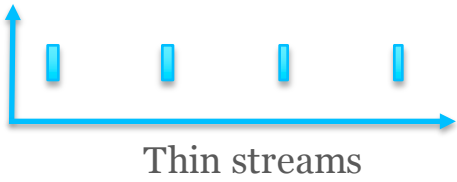


RDB: avoid retransmission delays





RDB: redundancy vs. latency gain



- Weigh redundancy against latency gain
- Protect against abuse
  - building a too high cwnd due to loss hiding

- Two key questions:
  1. When to bundle?
  2. How many redundant segments to allow?

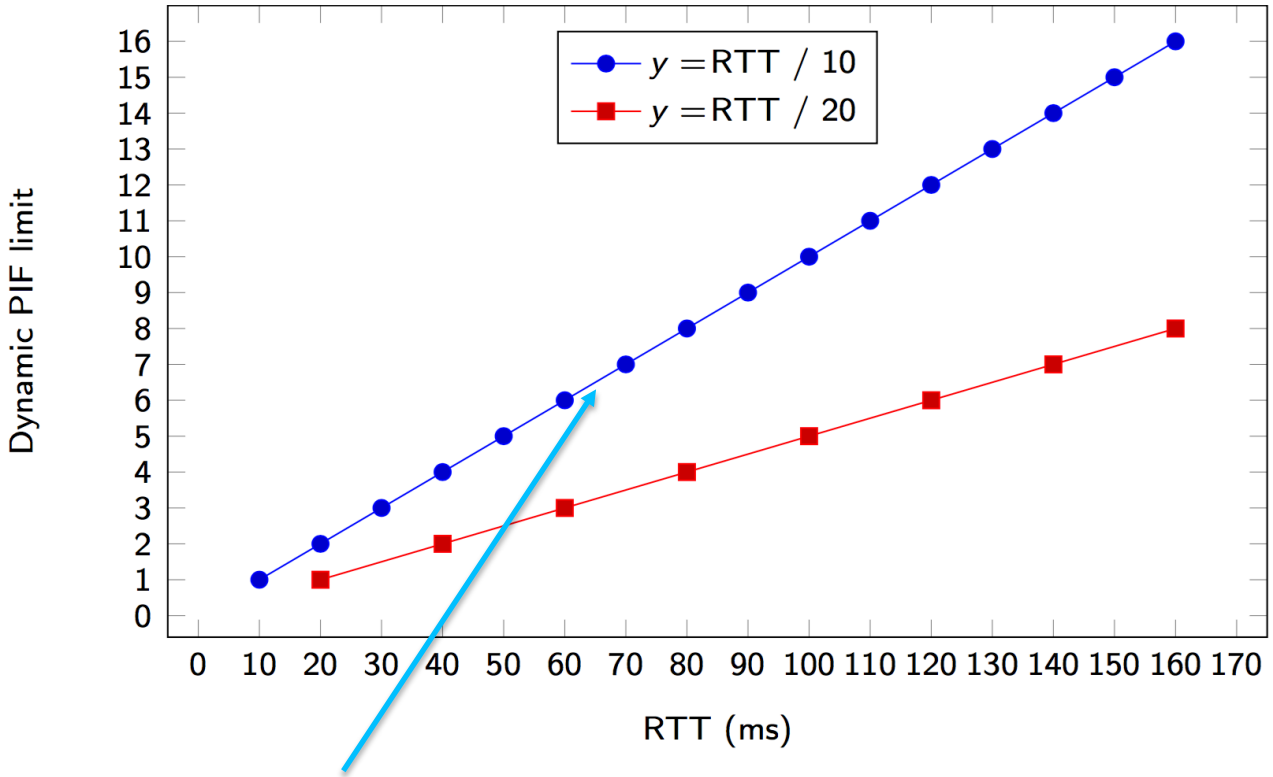
RDB: when to bundle?

- Use tcp_stream_is_thin (PIF < 4)?
  - big penalty for high-RTT flows.
  - a more precise name would be: can_trigger_fast_retransmit_within_one_rtt

```
/* Determines whether this is a thin stream (which may suffer from
 * increased latency). Used to trigger latency-reducing mechanisms.
 */
static inline bool tcp_stream_is_thin(struct tcp_sock *tp)
{
        return tp->packets_out < 4 && !tcp_in_initial_slowstart(tp);
}
```

RDB: Dynamic PIF limit

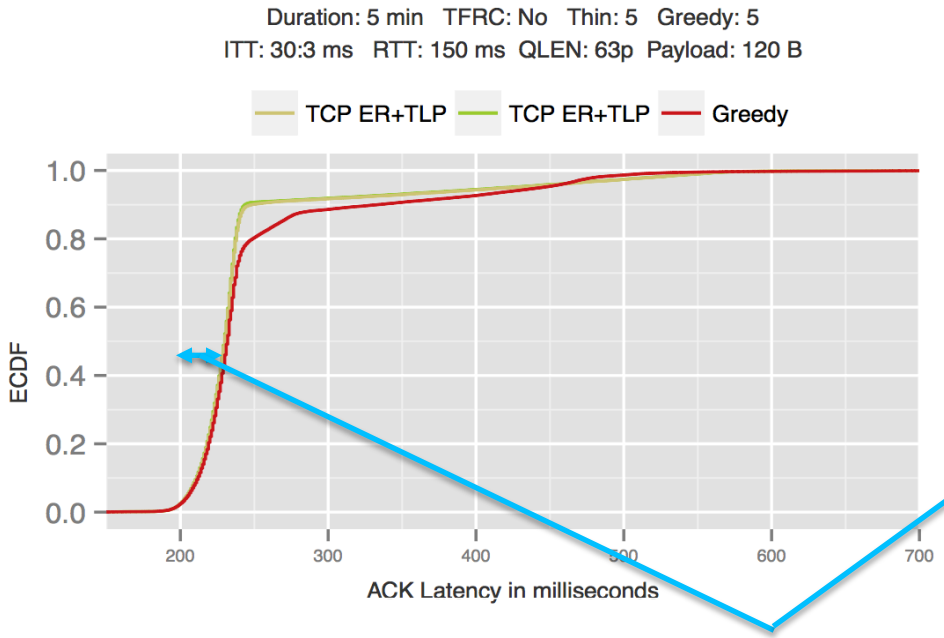
$$DPIFL = \frac{RTT_{min}}{ITT_{min}}$$

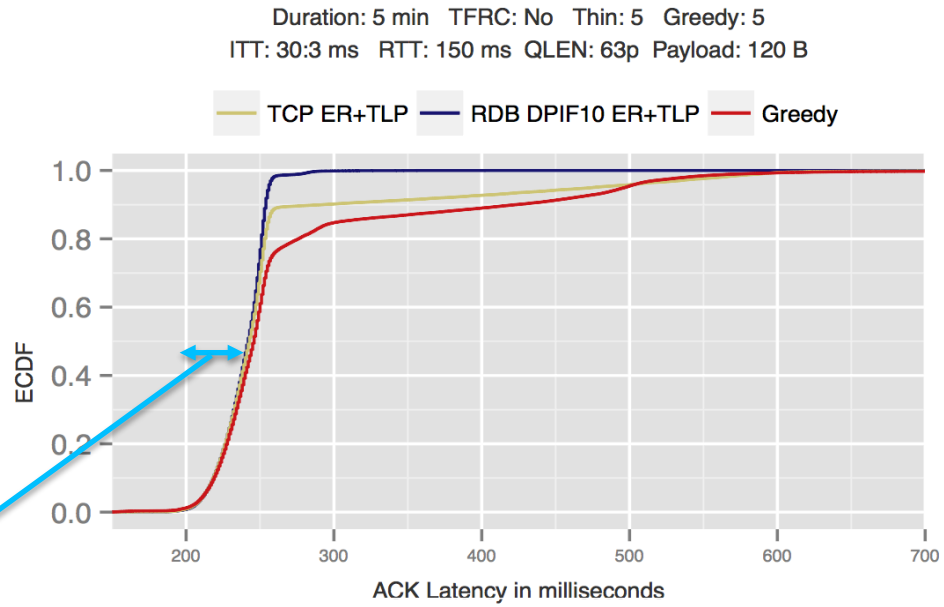


Choose TFRC-SP limit of 10ms ITT for thin streams [RFC4828].

RDB: how many segments?

- Avoid wantonly using capacity for redundancy.
- Recover faster from random loss.
- → allow RDB to only bundle only one segment.

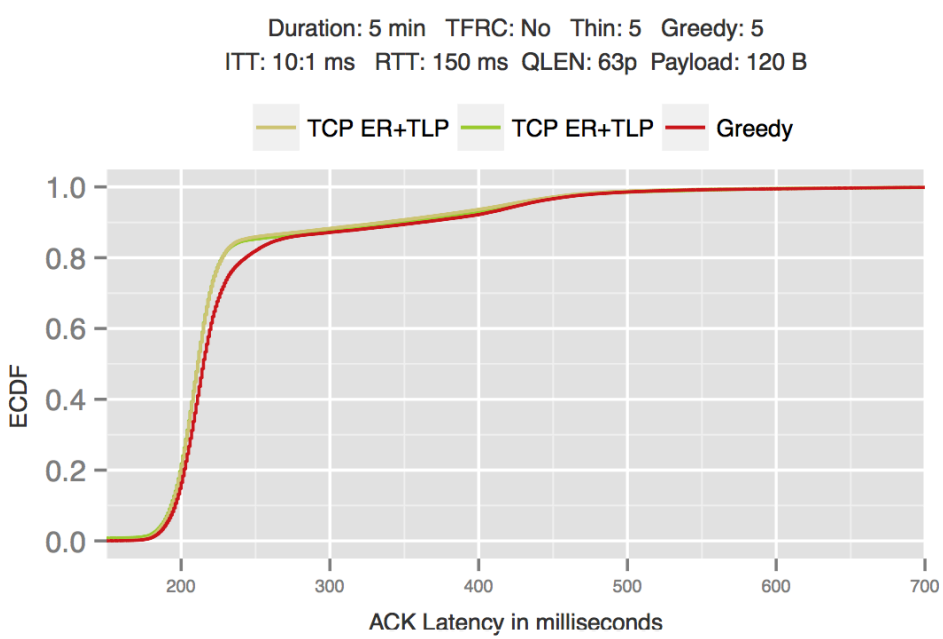


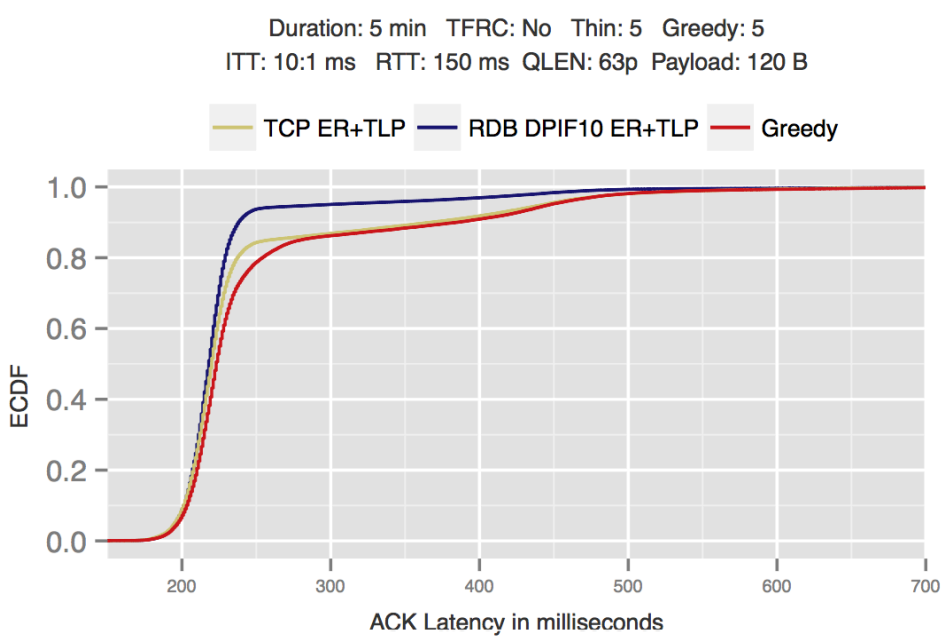


To bundle freely may contribute to added queueing delay
(an AQM would alleviate this situation)

RDB: how many segments?

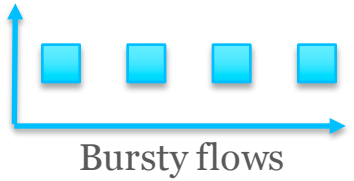
Thin streams





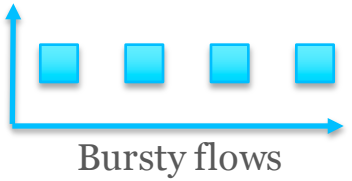
- a one segment limit helps avoid loss while keeping delay low.
- One may want to loosen the restrictions for special cases in order to further decrease latency.

New Congestion Window Validation (New-CWV)



- A method to control TCP cwnd for congestion-control in data-limited conditions
  - Data-limited do not consume the cwnd
  - Examples: interactive apps, web traffic, real-time flows
- Replaces RFC 2861 which was partially implemented in Linux
- It is defined in “draft-ietf-tcpm-newcwv” (TCPM WG item) approved by IETF for publication
- Implementations
  - For Linux (as a CC module and patch) http://github.com/rsecchi/newcwv
  - For FreeBSD https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=191520

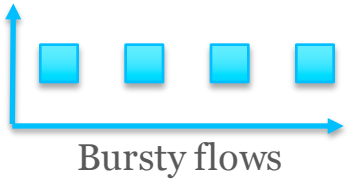
New-CWV Goals & Design



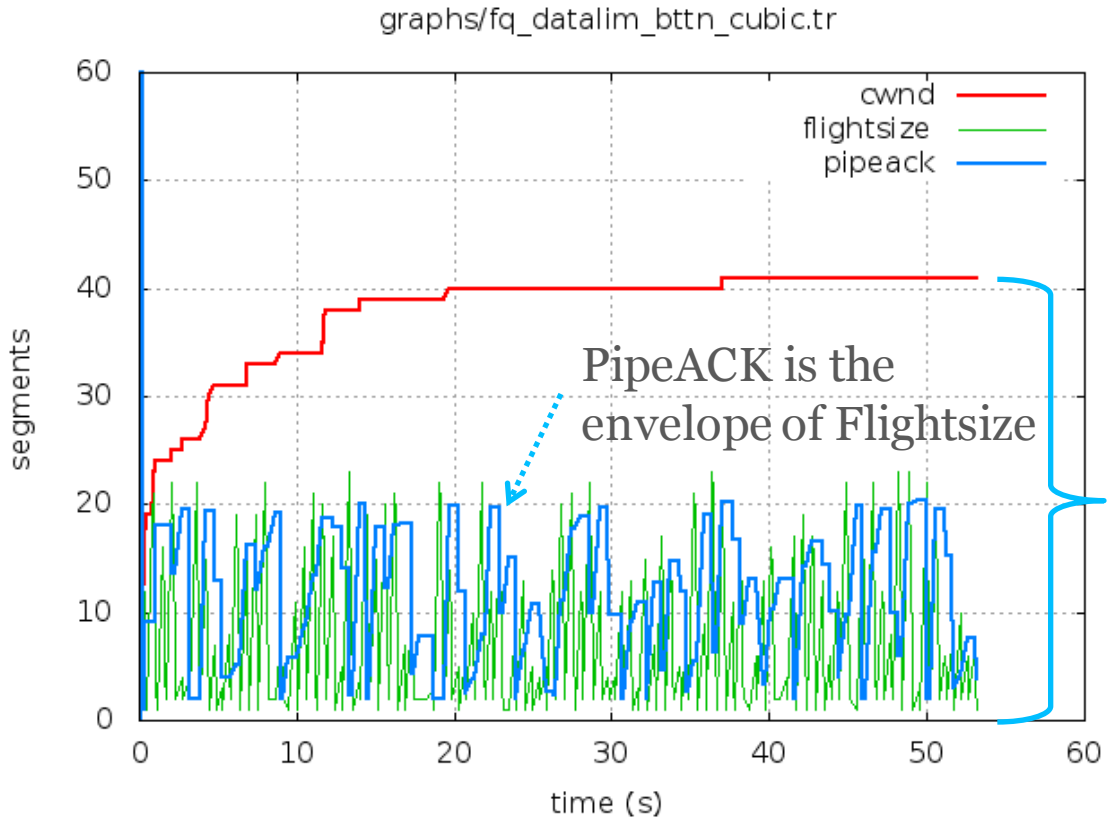
- New-CWV Goals
  - To reduce the latency in “bursty” applications
  - To remove the incentive for “ad-hoc” methods (eg “padding“)
  - To provide an incentive for the use of long-lived connections, rather than a succession of short-lived flows
  - To avoid a TCP sender growing a large "non-validated" cwnd (Linux did this already ☺)

- Design choices
  - TCP sender-side only modification
  - Change congestion control rules when the data to send is less than cwnd (non-validated periods)
  - Congestion control for data-limited flow independent from the RTT
  - Congestion control not based on the flightsize (it is not validated)

new-CWV method (1/3)

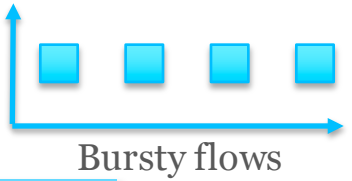


CC driven by available bandwidth evaluations (PipeACK)

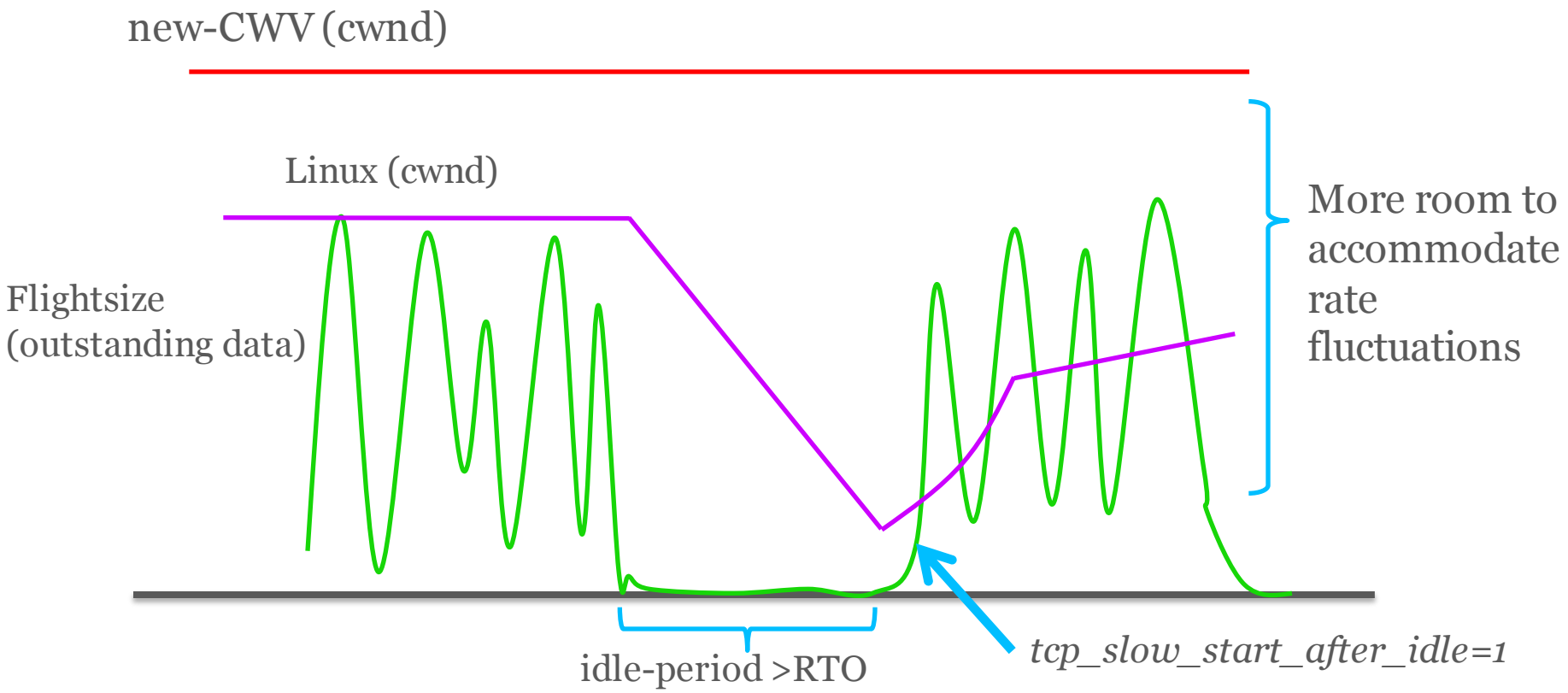


cwnd can grow up to 2 pipeACK

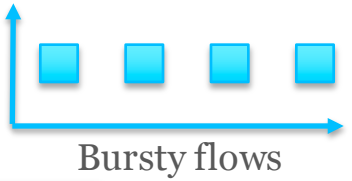
new-CWV method (2/3)



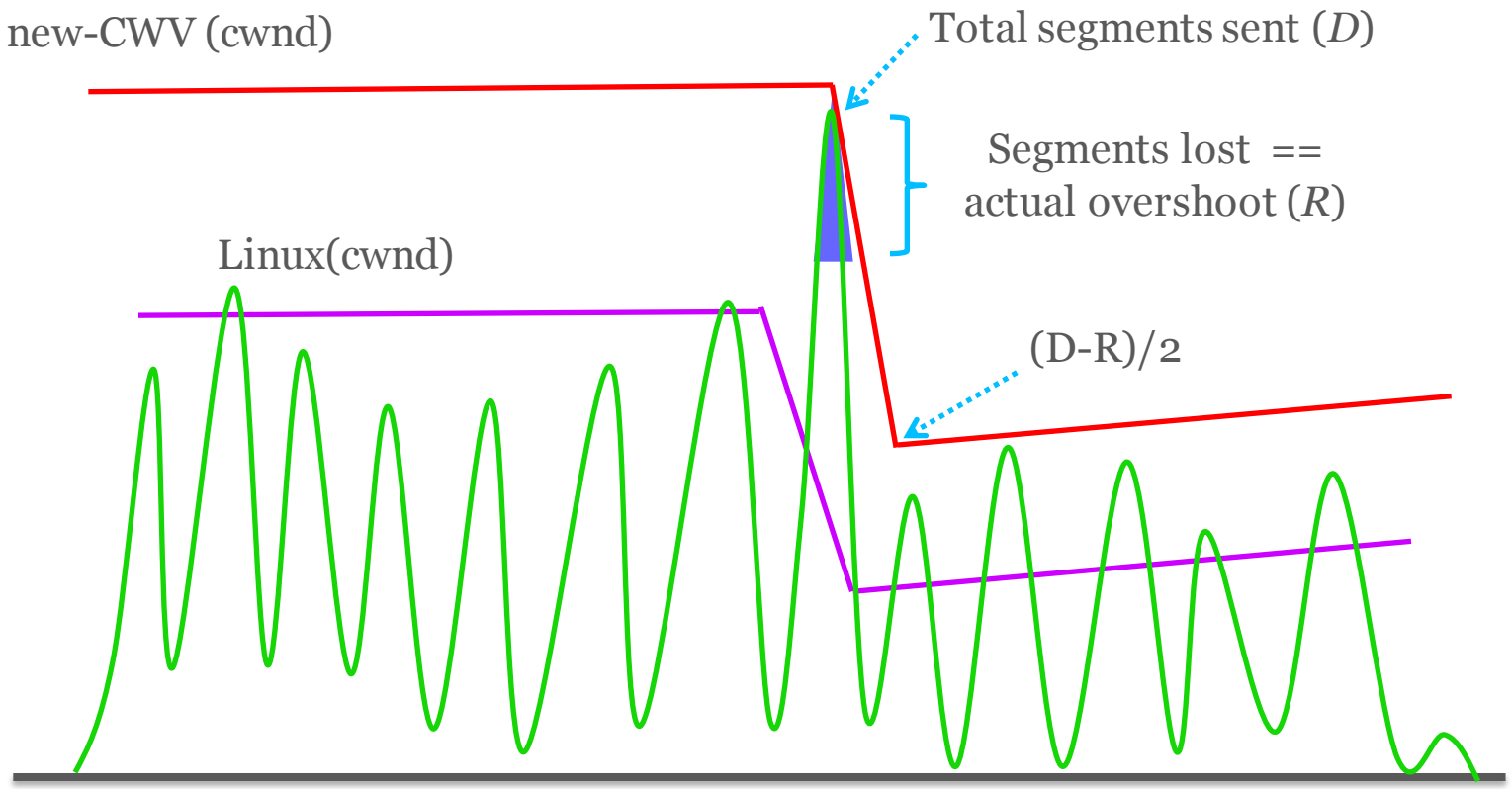
Avoid collapsing cwnd after periods > RTO if no congestion feedback



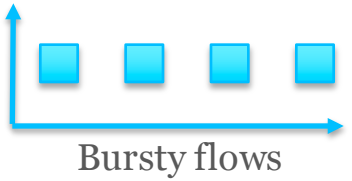
new-CWV method (3/3)



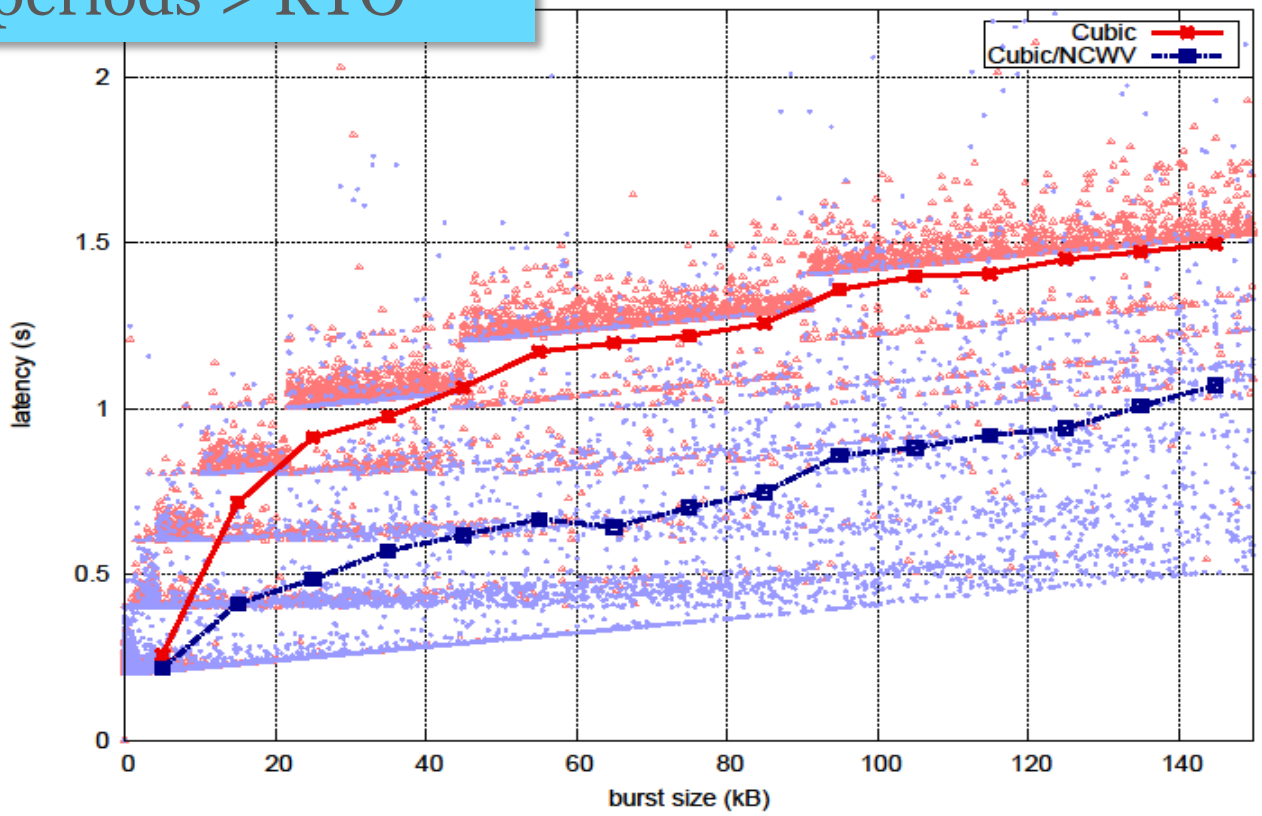
More accurate response to congestion feedbacks during data-limited periods



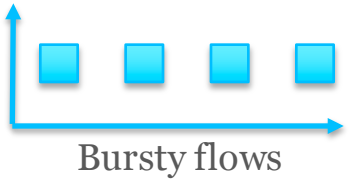
Experiments with TMIX replaying TCP connection from a wide-area traffic trace



Burst transmission latency after idle periods > RTO



Changes to Linux kernel



TCP headers: Socket descriptor & helper functions

include/linux/tcp.h

```
struct tcp_sock {
    [..]
    newcwv vars
};
```

New variables for the socket descriptor

```
static inline bool
tcp_is_cwnd_limited(sk)
{
   if ( flightsize>=cwnd ||
        pipeack>=cwnd) {
          return true;
   }
   return false;
}
```

Called by CC modules to determine if cwnd can be increased

TCP actions for outgoing packets:
Don't reduce cwnd

net/ipv4/tcp_output.c

```
tcp_event_data_sent(sk) {
   [...]
   tcp_newcwv_datalim_closedown(sk);
}
```

When packet are sent check if an RTO has passed from previous packet sent:

Reduce cwnd only after 5min rather than after one RTO as in CWV

CC common action: Initialise newcwv

net/ipv4/tcp_cong.c

```
tcp_init_congestion_control(sk) {
   [...]
   tcp_newcwv_reset(sk);
}
```

Init newcwv vars at start

Changes to Linux kernel: Recovery & PipeACK computation

net/ipv4/tcp_input.c

A **tcp_enter_recovery(sk)** {
[...]
if (pipeack<=2cwnd)

tcp_newcwv_enter_recovery(sk);
}

Loss detected (3dupacks):
1) store D
2) Cwnd = D/2

B **tcp_end_cwnd_reduction(sk)**
{
[...]
tcp_newcwv_end_recovery(sk);
tcp_newcwv_reset(sk);
}

At the end of recovery
1) cwnd=(D-R)/2
2) Restart newcwv

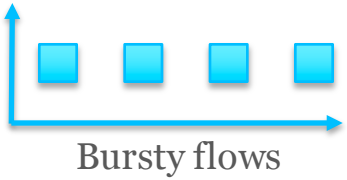
tcp_enter_loss(sk){
[...]
tcp_newcwv_reset(sk);
}

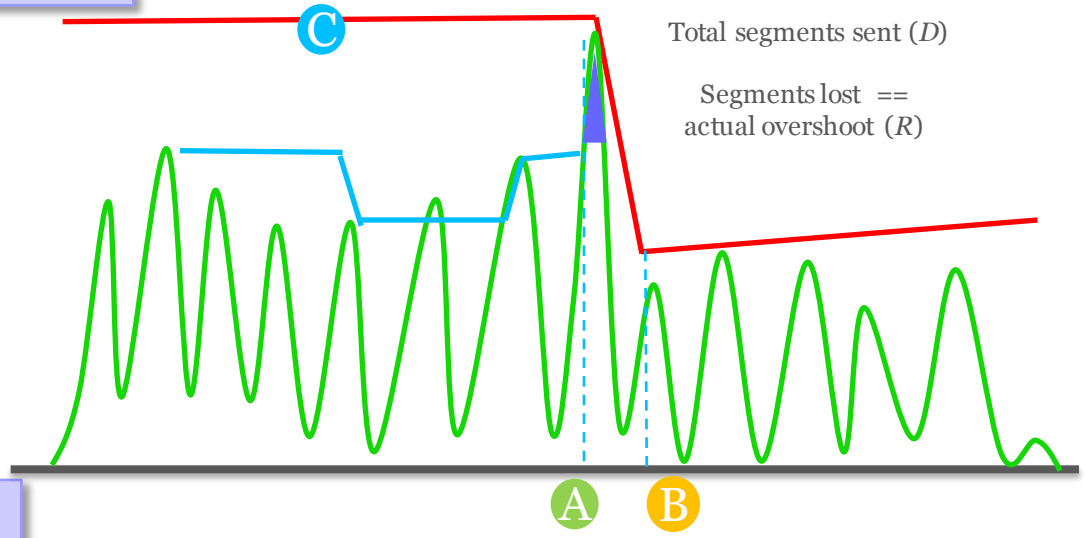
Timeout?
Restart newcwv

C **tcp_ack(sk)** {
[...]

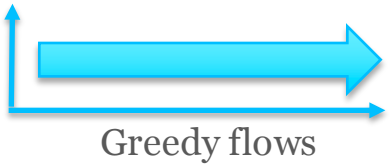
tcp_newcwv_update_pipeack(sk)
}

ACK received?
update
pipeACK





Caia Delay Gradient (CDG)



Developed by David Hayes / CAIA

Linux edition (by Kenneth Klette Jonassen)

Basic concepts

Delay-gradient as a congestion signal

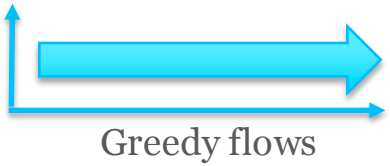
- RTT is a noisy signal
- RTTmin and RTTmax in a measured interval (1 RTT)

$$g_{\min,n} = \mathrm{RTT}_{\min,n} - \mathrm{RTT}_{\min,n-1}$$
$$g_{\max,n} = \mathrm{RTT}_{\max,n} - \mathrm{RTT}_{\max,n-1}$$

- Smoothed
  - moving average (configurable)
  - probabilistic back-off

- Queue State
  - $Q \in$ {full, empty, rising, falling, unknown}.

CDG: Delay-gradient probabilistic backoff



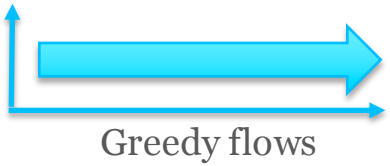
Why?

- Helps avoid synchronisation issues
- Helps smooth the noisy gradient signal.

Exponential

- Decisions are made once per RTT
- $P[\text{backoff}] = 1 - e^{-(\overline{g}_n/G)}$
- $\overline{P_{\text{RTT}}[\text{back-off}]} = \overline{P_{\text{RTT}}[\text{back-off}]}$
- Note: TCP's additive increase rate will still be RTT dependent.

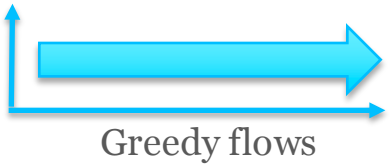
Co-existence with loss-based CC



- Maintains a “shadow congestion window”
  - switches to “New Reno” congestion avoidance when competing with loss-based CC flows.

- Also includes a loss-heuristic to detect random losses (not due to congestion).
  - disabled by default due to uncertainty to its accuracy (needs evaluation).

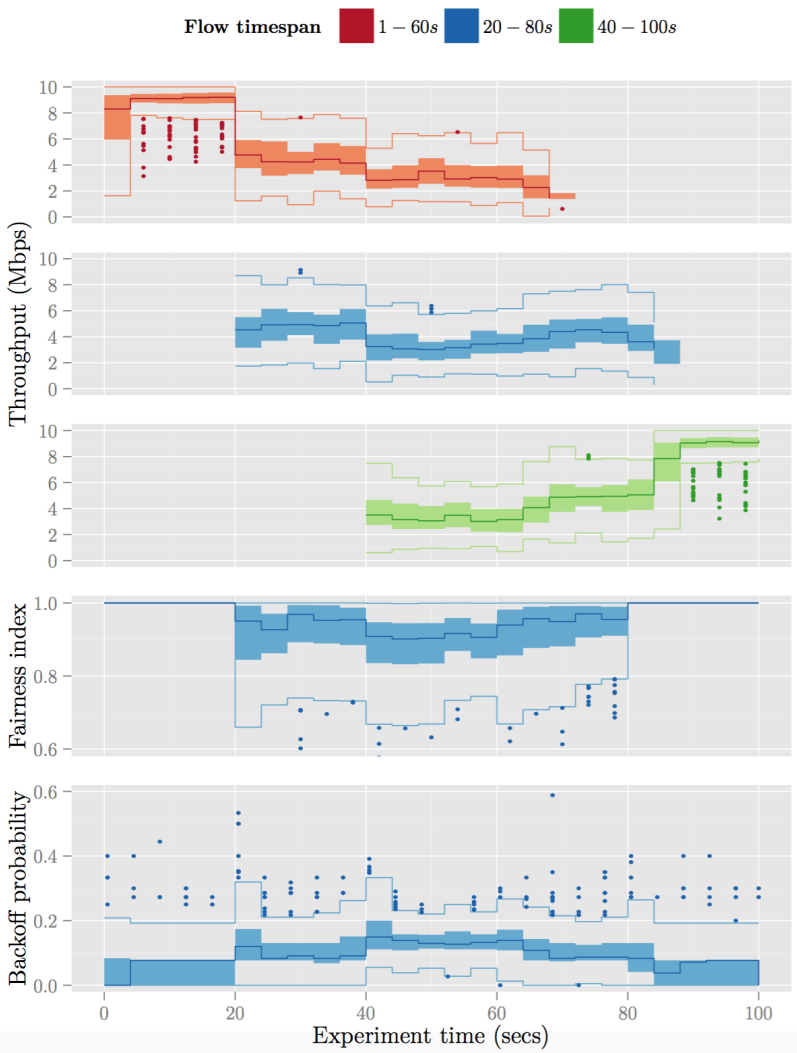
FreeBSD->Linux

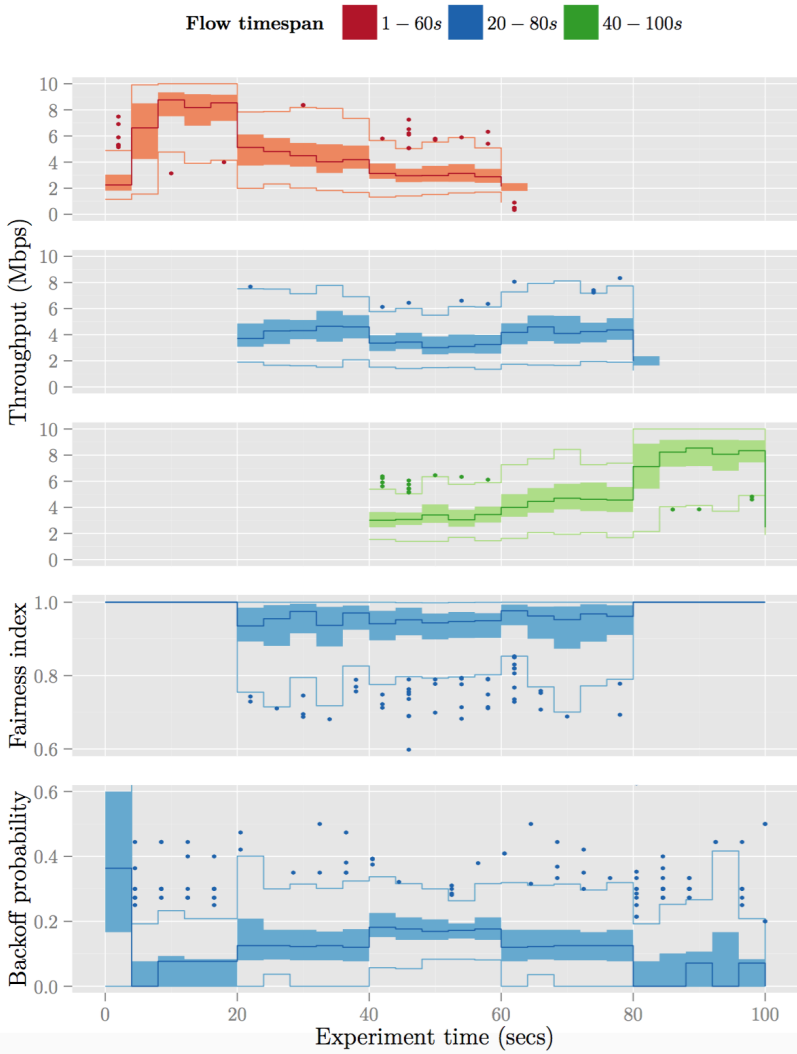


- CDG added to FreeBSD in 2013 (9.2).
- Some key differences in the Linux version by Kenneth:
  - Granularity of timers μsec in Linux, msec in FreeBSD
  - Using Hybrid Slow start and Proportional Rate Reduction.
  - Add toggle for shadow window mechanism. Suggested by David Hayes.
  - Add toggle for non-congestion loss tolerance.
  - Scaling parameter G is changed to a backoff factor;
    - conversion is given by: *backoff_factor* = 1000/(G * window).
  - Limit shadow window to 2 * cwnd, or to cwnd when application limited.
  - More accurate e^-x.
- CDG is available as a pluggable congestion control since 4.2

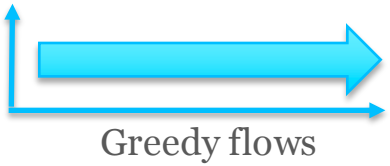
FreeBSD vs Linux performance

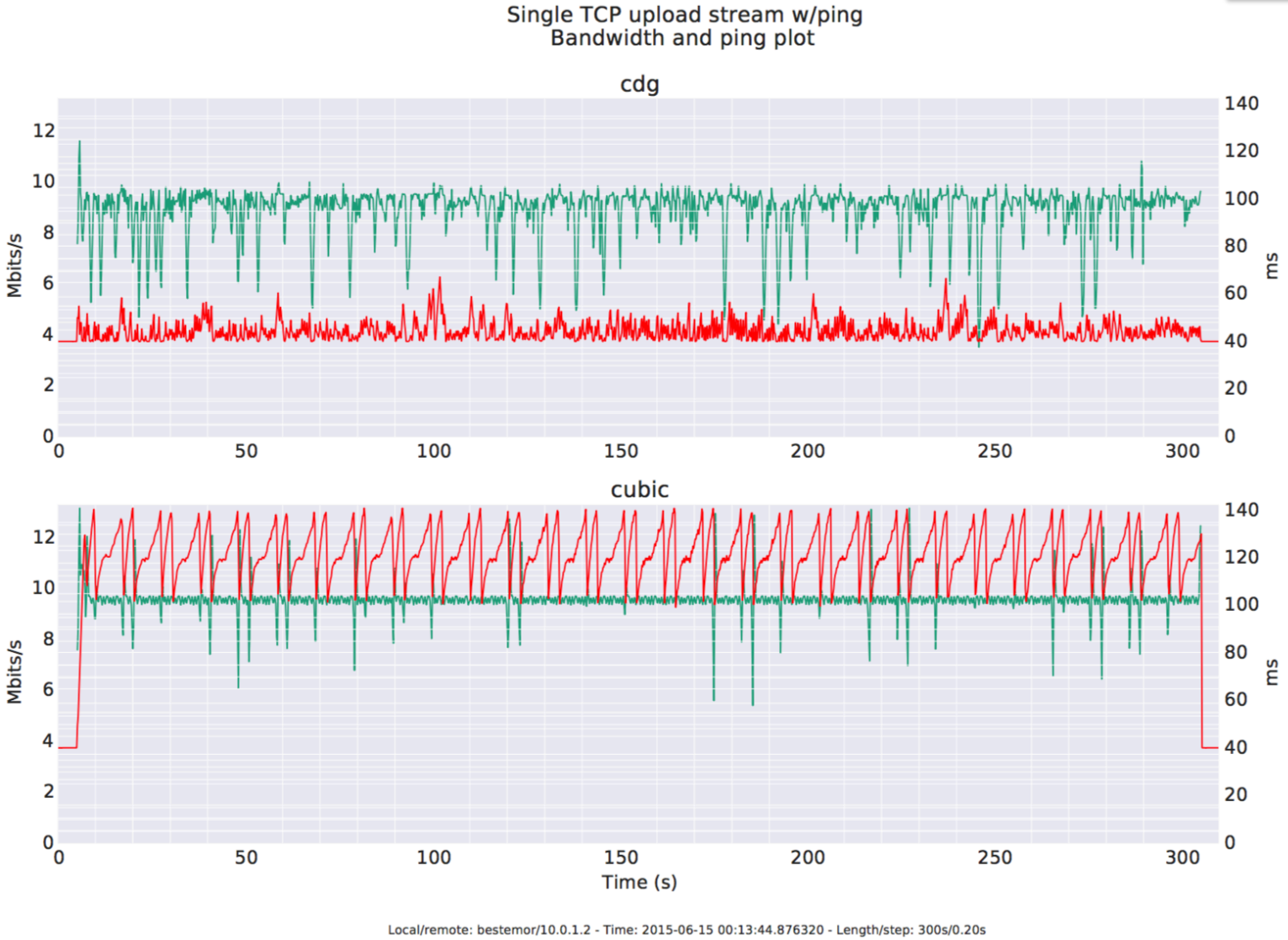
Greedy flows



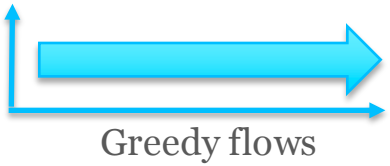


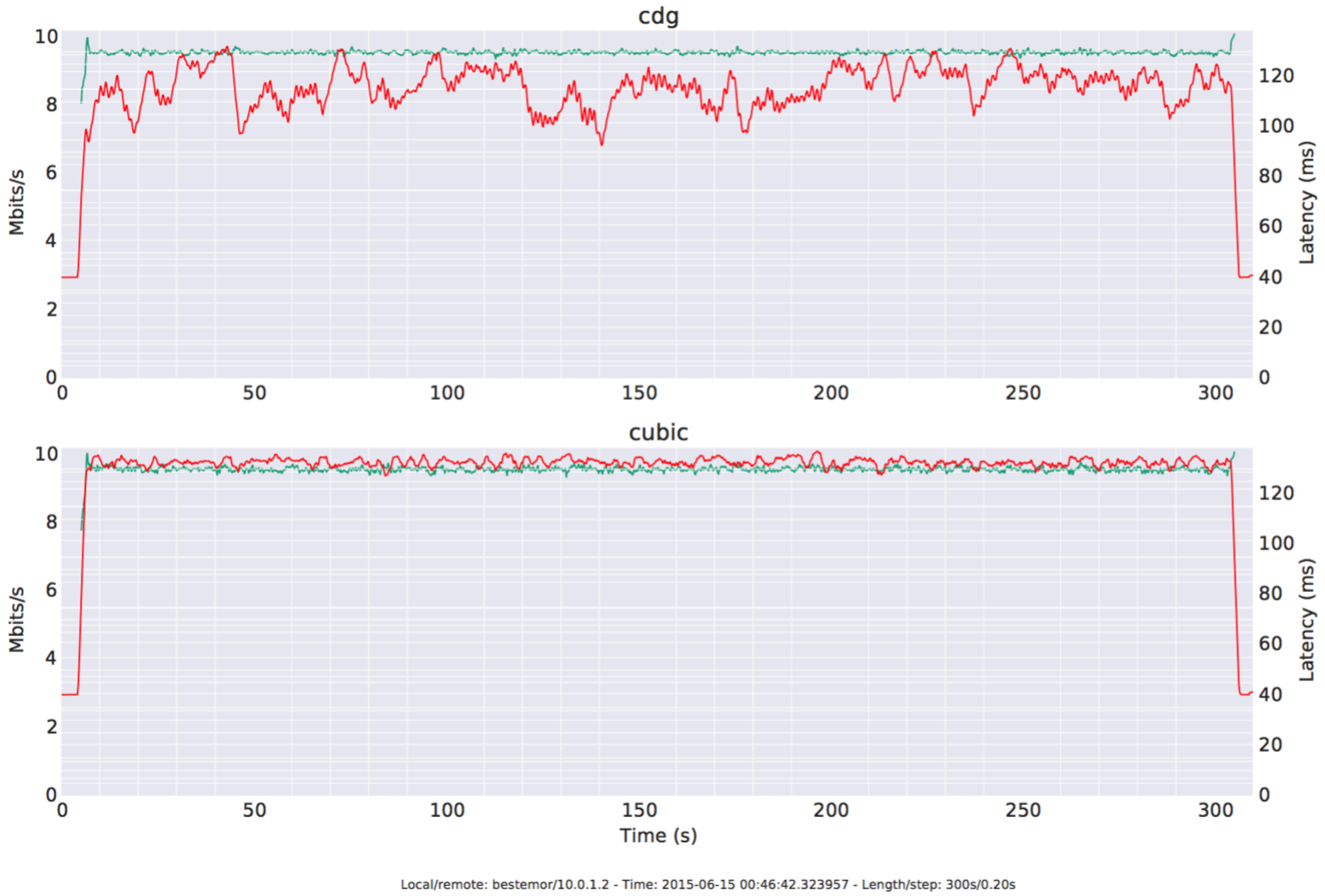
Delay compared to Cubic





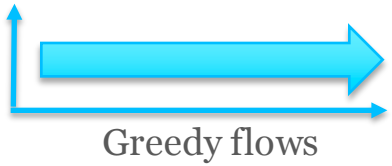
Delay compared to Cubic





L. Brakmo New Vegas tests

Loss competition turned on



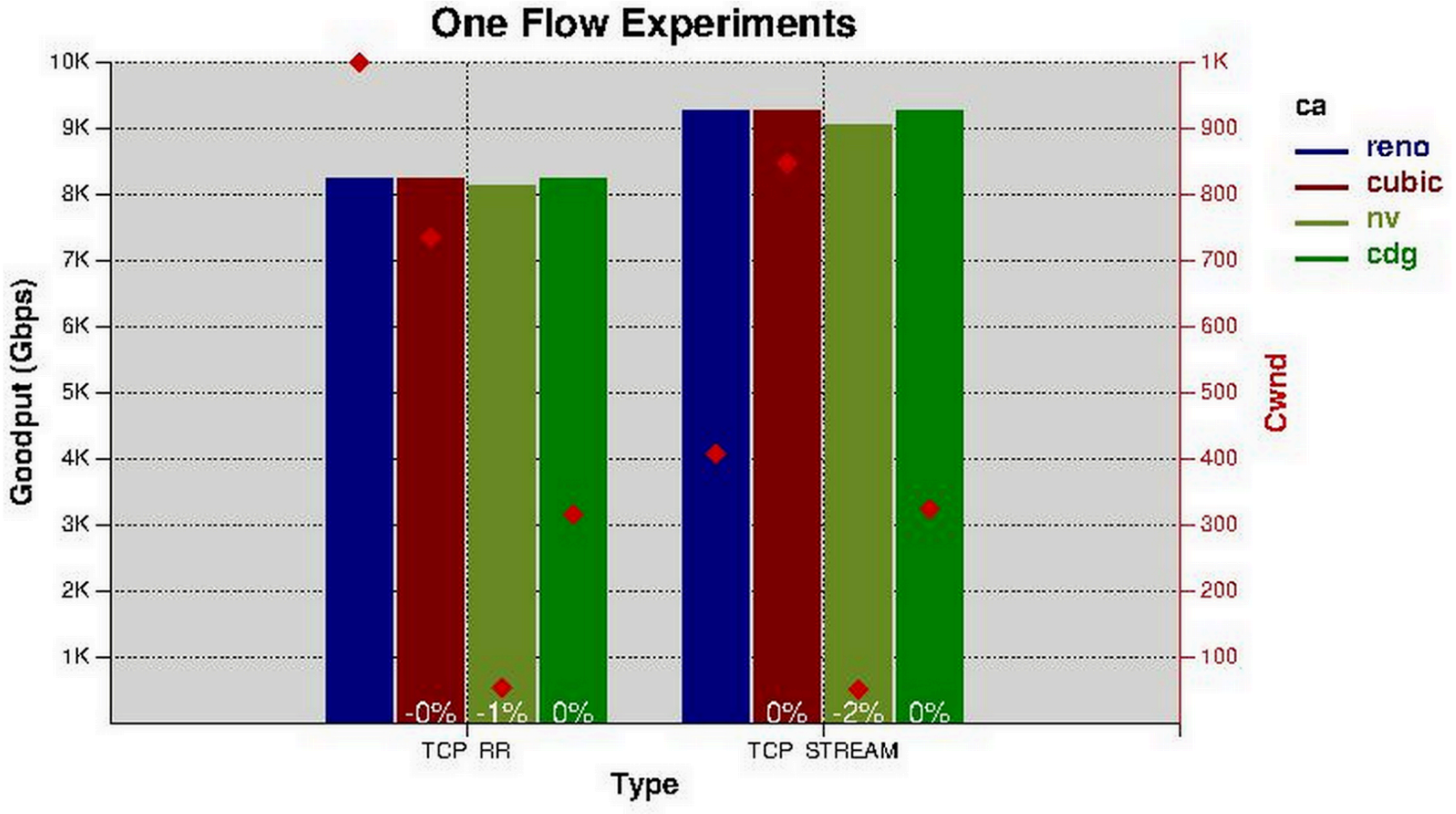


Chart 1: Rate and Cwnd for 1-flow experiments

L. Brakmo New Vegas tests: http://www.brakmo.org/networking/tcp-nv/TCPNV.html

References

- Internet latency: “Reducing Internet Latency: A Survey of Techniques and their Merits” by B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl. IEEE Communication Surveys and Tutorials.
- RTOR: M. Rajiullah et al, “An Evaluation of Tail Loss Recovery Mechanisms for TCP”, In ACM SIGCOMM CCR, Vol. 45(1), January 2015.
- TLP: T. Flach et al, “Reducing Web Latency: The Virtue of Gentle Aggression”, In ACM SIGCOMM CCR, Vol. 43(4), October 2013.
- RDB: Bendik Rønning Opstad “Taming Redundant Data Bundling - Balancing fairness and latency for redundant bundling in TCP”
- New CWV: Arjuna Sathiaseelan, Raffaello Secchi, Md. Israfil Biswas and Gorry Fairhurst, Enhancing TCP Performance to support Variable -Rate Traffic, ACM CoNext Capacity Sharing WorkShop (CSWS), Nice, December 2012.
- CDG: D.A. Hayes and G. Armitage. "Revisiting TCP congestion control using delay gradients." In Networking 2011, pages 328-341. Springer, 2011.
- CDG: K.K. Jonassen. "Implementing CAIA Delay-Gradient in Linux." MSc thesis. Department of Informatics, University of Oslo, 2015.
- L. Brakmo New Vegas tests (including CDG)
  - https://docs.google.com/document/d/1o-53jbO_xH-m9g2YCgjaf5bK8vePjWP6MkorYiRLK-U

Questions?

- Patches and more info:
  http://www.riteproject.eu/resources

Work funded by:

- The RITE EU-project: www.riteproject.eu
  - Project No.: ICT-317700
- "TimeIn" project - Research Council of Norway
  - Project No.: 213265