Reducing Latency for Linux Transport

Per Hurtig and Andreas Petlund

RITE – Reducing Internet Transport Latency

Dublin 06.10.2015
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)
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
  - avoiding queueing
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 Gradients (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

Receiver

Sender

RTO

Effective RTO
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
When restarting the RTO, set:

\[ \text{RTO} = \text{RTO} - T_{\text{earliest}} \]

where \( T_{\text{earliest}} \) 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

Reducing Internet Transport Latency

Baseline RTOR TLP TLPR

Normalized Flow Completion Time

RTT [ms]

delACKs

quickACKs

Short flows
Web Page Downloads

Reducing Internet Transport Latency

Web Page Downloads

Cumulative Density

OCT [ms]

Baseline
RTOR
TLP
TLPR

OCT [ms]

10ms RTT

160ms RTT
Changes to kernel (RTOR)

```c
void tcp_rearm_rto(struct sock *sk) {
    [...] else if (icsk->icsk_pending == ICSK_TIME_RETRANS &&
                   syscall_tcp_rto_restart &&
                   tp->packets_out < syscall_tcp_rto_restart &&
                   (tp->packets_out + tcp_unsent_pkts(sk) <
                    syscall_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);
}
net/ipv4/tcp_input.c
```

Short flows
Changes to kernel (TLPR)

```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

<table>
<thead>
<tr>
<th>Application</th>
<th>Payload size</th>
<th>Packet inter-arrival time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg</td>
<td>min</td>
</tr>
<tr>
<td>Windows Remote Desktop</td>
<td>111</td>
<td>8</td>
</tr>
<tr>
<td>VNC (from client)</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>VNC (from server)</td>
<td>827</td>
<td>2</td>
</tr>
<tr>
<td>Skype (2 users)</td>
<td>236</td>
<td>14</td>
</tr>
<tr>
<td>SSH text session</td>
<td>48</td>
<td>16</td>
</tr>
<tr>
<td>Anarchy Online</td>
<td>98</td>
<td>8</td>
</tr>
<tr>
<td>World of Warcraft</td>
<td>26</td>
<td>6</td>
</tr>
<tr>
<td>Age of Conan</td>
<td>80</td>
<td>5</td>
</tr>
</tbody>
</table>
**RDB – main principle**

- 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

Time | Segment
--- | ---
seq=0  
payload=200

seq=200  
payload=200

seq=200  
payload=400

seq=200  
payload=600

seq=800  
payload=200

Sender

Receiver

Thin streams

ack=200

ack=800

ack=1000

RTT

ITT

seq=200

payload=200

seq=800

payload=200

seq=1000

payload=200

seq=1200

payload=200

seq=1400

payload=200

seq=1600

payload=200
RDB: results

- High-loss example
  - uniform loss (netem)
  - uo competing traffic -> minimal queueing delay
  - static PIF limit
RDB: results

High-loss example

Duration: 5 min  Streams: 20  RTT: 150 ms  QLEN: 60
Loss: 10%  ITT: 30.3 ms

ECDF

ACK Latency in milliseconds

<table>
<thead>
<tr>
<th>Name</th>
<th>Mean</th>
<th>40th</th>
<th>50th</th>
<th>60th</th>
<th>70th</th>
<th>80th</th>
<th>90th</th>
<th>95th</th>
<th>97th</th>
<th>99th</th>
<th>99.9th</th>
<th>99.99th</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>270</td>
<td>150</td>
<td>150</td>
<td>205</td>
<td>300</td>
<td>385</td>
<td>480</td>
<td>654</td>
<td>775</td>
<td>956</td>
<td>2150</td>
<td>2852</td>
<td>4095</td>
</tr>
<tr>
<td>RDB 4</td>
<td>227</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>240</td>
<td>318</td>
<td>389</td>
<td>463</td>
<td>551</td>
<td>717</td>
<td>1511</td>
<td>5495</td>
<td>9650</td>
</tr>
<tr>
<td>RDB 8</td>
<td>154</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>180</td>
<td>183</td>
<td>209</td>
<td>250</td>
<td>813</td>
<td>3154</td>
<td></td>
</tr>
</tbody>
</table>
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`

```c
/* 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

\[ DPIFL = \frac{RTT_{\text{min}}}{ITT_{\text{min}}} \]

Choose TFRC-SP limit of 10ms ITT for thin streams [RFC4828].
RDB: how many segments?

- Want to avoid wantonly using capacity for redundancy.
- Want to fix random losses.
- → allow RDB to only bundle only one segment.
No limit for the allowed number of segments
To bundle may contribute to added queueing delay
RDB: how many segments?

- A one segment limit helps avoid random loss while keeping delay minimal.
- May 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 (e.g., “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)

PipeACK is the envelope of Flightsize

cwnd can grow up to 2 pipeACK
new-CWV method (2/3)

Avoid collapsing cwnd after periods > RTO if no congestion feedback

new-CWV (cwnd)

Linux (cwnd)

Flightsize (outstanding data)

idle-period > RTO

tcp_slow_start_after_idle = 1

More room to accommodate rate fluctuations

Bursty flows
new-CWV method (3/3)

More accurate response to congestion feedbacks during data-limited periods

- Total segments sent \((D)\)
- Segments lost == actual overshoot \((R)\)
- \((D-R)/2\)
Experiments with TMIX replaying TCP connection from a wide-area traffic trace

Bursty flows

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
};

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

New variables for the socket descriptor

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

net/ipv4/tcp_cong.c

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

Init newcwv vars at start

Bursty flows

TCP actions for outgoing packets:

New variables for the socket descriptor

Called by CC modules to determine if cwnd can be increased

Reduced Internet Transport Latency
Changes to Linux kernel: Recovery & PipeACK computation

net/ipv4/tcp_input.c

```c
A tcp_enter_recovery(sk) {
    [...]
    if (pipeack<=2cwnd)
        tcp_newcwavv_enter_recovery(sk);
}

B tcp_end_cwnd_reduction(sk) {
    [...]
    tcp_newcwavv_end_recovery(sk);
    tcp_newcwavv_reset(sk);
}

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

tcp_ack(sk) {
    [...]
    tcp_newcwavv_update_pipeack(sk)
}
```

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

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

Timeout?
Restart newcwavv

ACK received?
update pipeACK

Bursty flows

Total segments sent (D)
Segments lost == actual overshoot (R)

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

\[
\begin{align*}
g_{\text{min},n} &= \text{RTT}_{\text{min},n} - \text{RTT}_{\text{min},n-1} \\
g_{\text{max},n} &= \text{RTT}_{\text{max},n} - \text{RTT}_{\text{max},n-1}
\end{align*}
\]

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

- Queue State
  - \( Q \in \{\text{full, empty, rising, falling, unknown}\}. \)
Delay-gradient probabilistic backoff

Why?

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

Exponential

- Decisions are made once per RTT
- $P_{\text{backoff}} = 1 - e^{-(gn/G)}$
- $P_{\text{RTT (back-off)}} = P_{\text{RTT (back-off)}}$
- 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 aiming to detect random losses (not due to congestion).
  - Disabled by default due to uncertainty to its accuracy (needs evaluation).
CDG added to FreeBSD in 2013 (9.2).

Some key differences in the Linux version by Kenneth:

- Granularity of timers usec 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.
FreeBSD vs Linux performance

Linux CDG
\[ G = 3, \quad Delay = 2 \times 35 \text{ ms.} \]

Flow timespan: 1 – 60s, 20 – 80s, 40 – 100s

Throughput (Mbps)

Fairness index

Backoff probability

Experiment time (secs)

FreeBSD CDG
\[ G = 3, \quad Delay = 2 \times 35 \text{ ms.} \]

Flow timespan: 1 – 60s, 20 – 80s, 40 – 100s

Throughput (Mbps)

Fairness index

Backoff probability

Experiment time (secs)

Greedy flows
Delay compared to Cubic

Single TCP upload stream w/ping
Bandwidth and ping plot

cdg

cubic

Greedy flows

Local/remote: bestemor/10.0.1.2 - Time: 2015-06-15 00:13:44.876320 - Length/step: 300s/0.20s
Delay compared to Cubic

8 down - dslreports dsl test equivalent
Total bandwidth and average ping plot

Greedy flows
L. Brakmo New Vegas tests
Loss competition heuristics turned on.

Chart 2: 10KB vs. 1MB RPC Latencies

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

- RDB: Bendik Rønning Opstad “Taming Redundant Data Bundling - Balancing fairness and latency for redundant bundling in TCP”
- L. Brakmo New Vegas tests (including CDG)
  • https://docs.google.com/document/d/1o-53jbO_xH-m9g2YCjaf5bK8vePjWP6MkorYiRLK-U/edit

http://www.riteproject.eu/