TCP Enhancements in Linux

Pasi Sarolahti
Berkeley Summer School
6.6.2002

Outline

- TCP details per IETF RFC's
- Pitfalls in the specifications
- Linux TCP congestion control engine
- Features
- Discussion on performance
- Aside from Linux: F-RTO
- Conclusions

TCP Basics

- Slow start, congestion avoidance
- Receiver generates duplicate ACKs when data is missing
- Fast retransmit at third duplicate ACK
- Fast recovery to keep the ”ACK clock” in pace
  - Standard Reno (RFC 2581) or NewReno (RFC 2582)
- Without SACK at most one retransmission in RTT
- Retransmission Timer adjusted smoothly based on measured round-trip times
  - SRTT + 4 * RTTVAR

Some TCP Enhancements

- SACK: allow several retransmissions in RTT
  - acknowledge separate blocks of received data
  - conservative: ”holes” are still outstanding
  - Forward ACKs (FACK): ”holes” are considered lost
- D-SACK: report duplicate segments using SACK
- Timestamps: measure RTT for retransmissions
- Eifel: report unnecesary retransmissions using timestamps
- ECN: Explicit Congestion Notification
- Limited transmit: Avoid timeouts with small window

Discussion on Specifications

- RFC 2581 & RFC 2582: Congestion Control
  - Cwnd is artificially increased on duplicate ACKs. It does not correspond to real number of segments allowed to be in flight

in flight = SND.NXT – SND.UNA

- SACK congestion control draft
  - Separate document that assumes SACK is in use
  - Cwnd is not artificially increased
  - We need to implement both? Nah...

in flight = SND.NXT – SND.UNA – SACKed

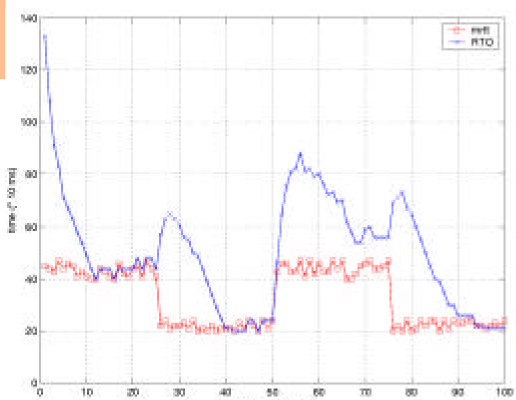
- RFC 2988 does not work well with high-granularity timers
  - No one sees this, because RTTs are generally below 1000ms

RFC 2988: RTO Calculation

RTTVAR <- ¾ * RTTVAR + ¼ * | SRTT – MRTT |

SRTT <- 7/8 * SRTT + 1/8 * MRTT

RTO <- max(1000ms, SRTT + 4 * RTTVAR)



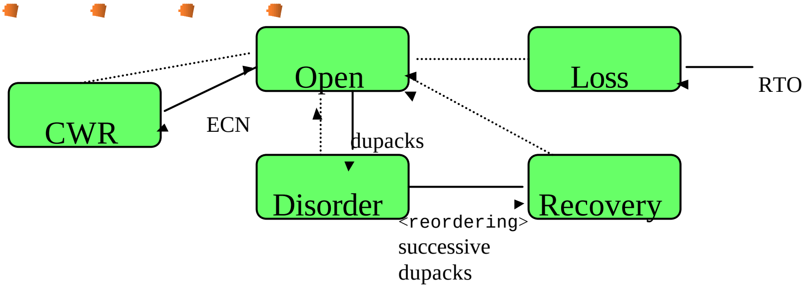
- RTO estimator decays rapidly
- When measured RTT drops, RTO goes up
- No one cares, because
  - Min limit of 1000ms
  - Coarse-grain timers

Linux Approach

in flight = packets_out – sacked_out – lost_out + retrans_out

- Common congestion control with Reno, SACK, FACK
- *sacked_out:* # of segments surely left network
  - SACK: number of SACKed segments
  - Reno: number of duplicate ACKs
- *lost_out*: # of segments suspected lost
  - SACK & Reno: first unacknowledged is considered lost
  - FACK: holes between SACKs are considered lost
- scoreboard markings are updated accordingly

CA States



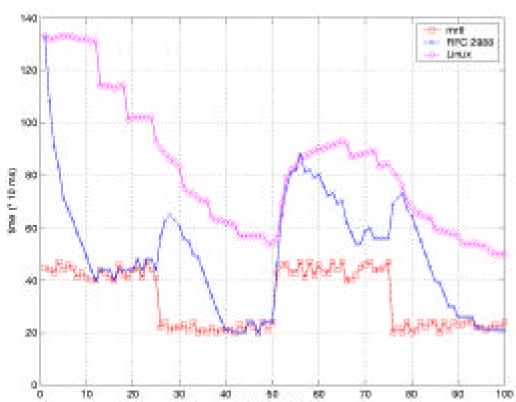
- `<reordering>` is adjusted when unnecessary retransmission is detected
  - by default 3
- Window is increased in *Open* and *Loss* states
- Window is decreased in *CWR* and *Recovery* states

Features

- Implements Explicit Congestion Notification (ECN)
- Congestion window is decreased steadily every second ACK in *CWR* and *Recovery* states
  - as in "rate-halving"
- Disorder state implements "Limited transmit" in practice
- Congestion window validation: If congestion window is not fully used for a while, it is reduced
- Congestion control state is cached for future connections

Linux Retransmission Timer

- Based on RFC 2988
- min. RTO = 200 ms
- min. RTTVAR = 50 ms
- RTTVAR reduced once per round-trip time
  - but increased instantly
- if RTT drops significantly, RTTVAR weight is reduced to 1/32



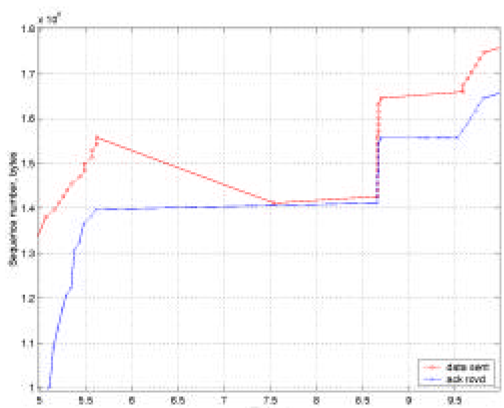
Congestion Window Undoing

- TCP sender can make false retransmits, e.g. due to
  - false RTOs caused by unexpected delay
  - dupacks caused by reordering in network
- False retransmits can be detected by using
  - TCP timestamps: receiver echoes timestamp of original segment after retransmission
  - D-SACKs: a retransmitted segment is acknowledged in cumulative ACK and in D-SACK
- After detecting false retransmission the sender sets
  - cwnd <- max(cwnd, ssthresh * 2)
  - ssthresh <- prior_ssthresh

Undoing on TCP Timestamps

<u>Without timestamps</u>

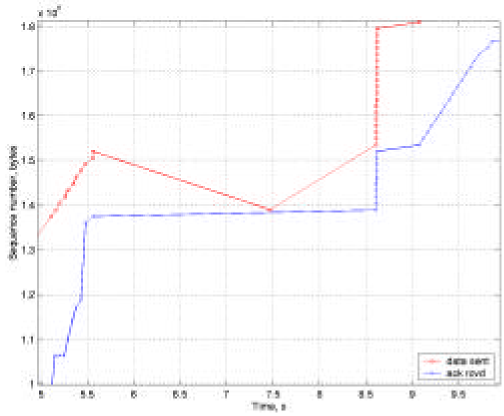
- A 3-second excessive delay occurs on 256Kbps link
- Triggers RTO, but ACKs for original segments arrive after RTO
- congestion window is halved
- 65 KB acknowledged between 5 and 10 s.



Undoing on TCP Timestamps

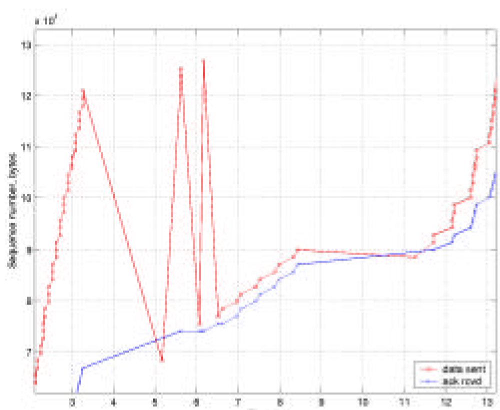
<u>**With timestamps**</u>

- Next ACK after RTO echoes timestamp of original segment
- Spurious timeout is detected
  - continue by transmitting new data
  - revert recent changes on congestion control parameters
- 75 KB acknowledged between 5 and 10 s.



Undoing Can Fail

- Link outage: One window of data segmenents and ACKs are dropped
- ACKs echo latest timestamp that updated window
- Because ACKs are lost, sender thinks new ACK acknowledged earlier data
  - Declares RTO spurious



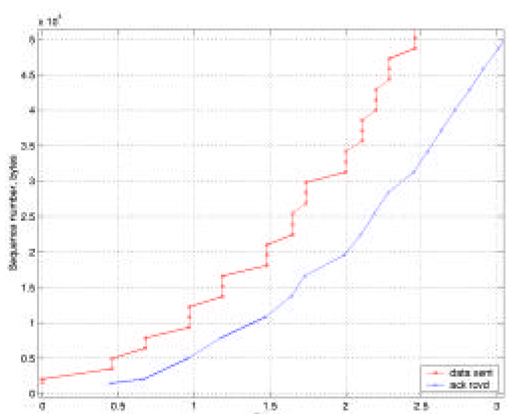
Delayed Acknowledgements

- Delayed acknowledgements should be used to avoid silly window syndrome
- Linux receiver measures interarrival times and adjusts delay timer accordingly
  - goal is to get an ACK out for every second segment
- Quick acknowledgements can be used at the beginning of the connection
  - causes the sender to increase the window faster
  - to avoid SWS, no more than (advwin / 2) quick acknowledgements are allowed

Effect of Quick Acks

<u>Without quickacks</u>

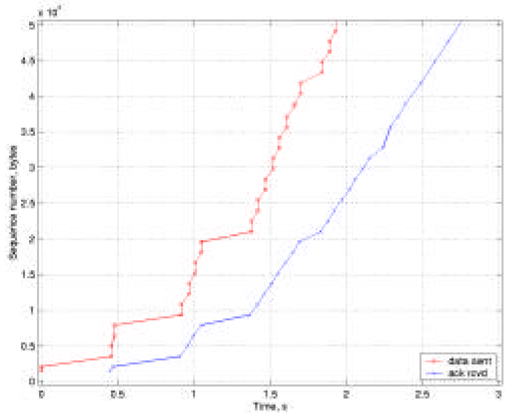
- 256 Kbps, 200 ms delay => BW*delay more than 12 KB
- 4-5 round-trips until the link is fully utilized
- every second segment is acknowledged
- 50 KB transmitted in 2.5 seconds



Effect of Quick Acks

With quickacks

- For the first 32 KB every segment is acknowledged
- 50 KB transmitted in 2 seconds



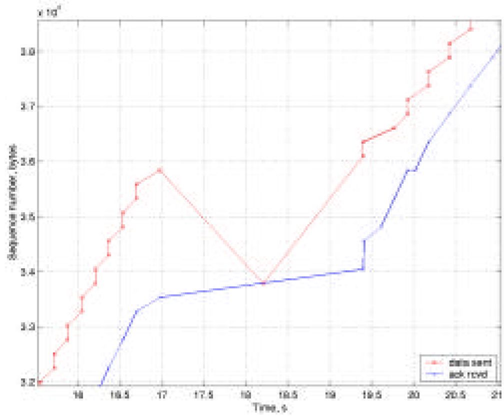
F-RTO

- Why should we retransmit everything after RTO?
- Transmit two new segments after the RTO
- If the resulthing two ACKs advance the window, we have a suspected spurious timeout in our hands
- If they don't advance the window, reset cwnd to 1 + RTT's after RTO = 3, and retransmit unacknowledged
- No need for SACK or timestamps
- F-RTO is _not_ about congestion window undoing
- ...but works well together with Eifel or D-SACK

F-RTO Behaviour

Delay on the link

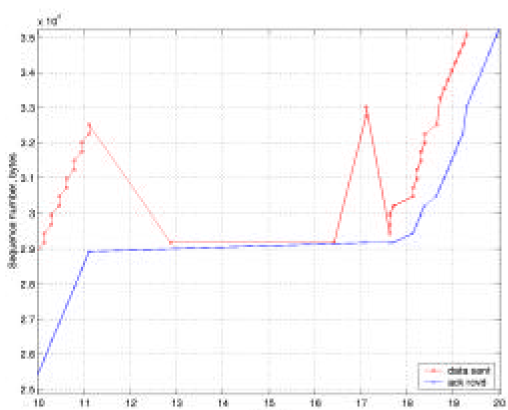
- On RTO the first segment is retransmitted
- Next two ACKs advance window => continue by transmitting new data
- At least the second ACK was for delayed segment
- Congestion window is reduced to half due to RTO



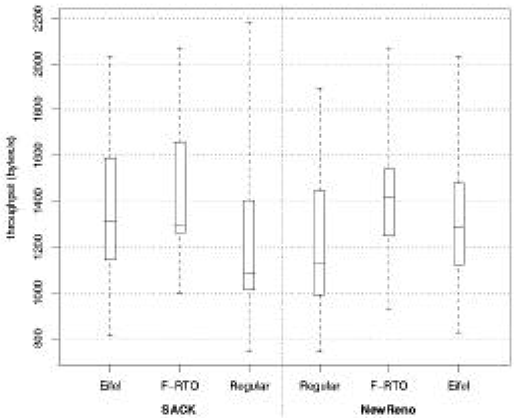
F-RTO Behaviour

Burst error on the link

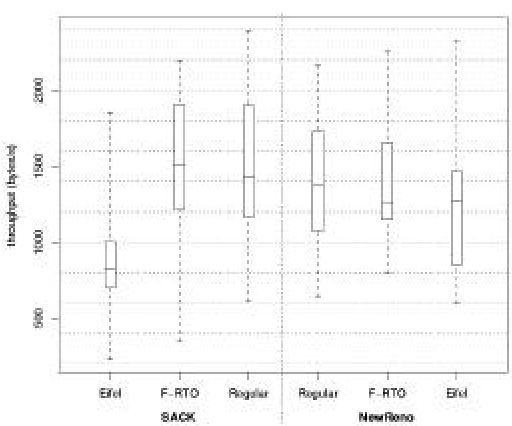
- First ACK advances the window => transmit two new segments
- Second ACK does not => start retransmitting in slow start



Performace with Delays



Performace with Burst Errors



Concluding Remarks

- Implementation follows packet conservation in practice
  - congestion window always holds a valid value
  - counters try to estimate how many packets really are outstanding
- If the data structures tracking outstanding and supspected losses are incorrect they are corrected, if incorrectness is detected
- Retransmission timer tries to avoid the pitfalls of the original algorithm