Linux TCP

Pasi Sarolahti

Seminar on Linux Kernel

12.12.2002

Outline

- Basic TCP concepts
- Linux TCP basics (wrt. BSD Unix)
- Linux TCP Retransmission Engine
- Linux TCP Features
- Conclusion

Basic TCP Concepts

- Reliable byte streams, data is delivered in order
- Congestion control limits the transmission rate
  - Slow start *(when cwnd < ssthresh)*
  - Congestion avoidance *(when cwnd >= ssthresh)*
  - Congestion window is reduced after packet loss
- Different loss recovery variants
  - Three successive duplicate ACKs are a signal of data loss
  - Old recovery algorithms (e.g. Reno) used in earlier Windows hosts and some BSDs
  - NewReno improves performance when multiple packet losses occur in same window
  - SACK option allows acknowledgement of discontinuous blocks of data (better performance with multiple packet losses
- Retransmission timer
  - Dynamically adjusted, based on measured packet round-trip times
  - When timer expires, unacknowledged segments are retransmitted by default

BSD TCP vs. Linux TCP

By tradition, TCP specs and books are often based on the concepts used in BSD Unix systems

BSD

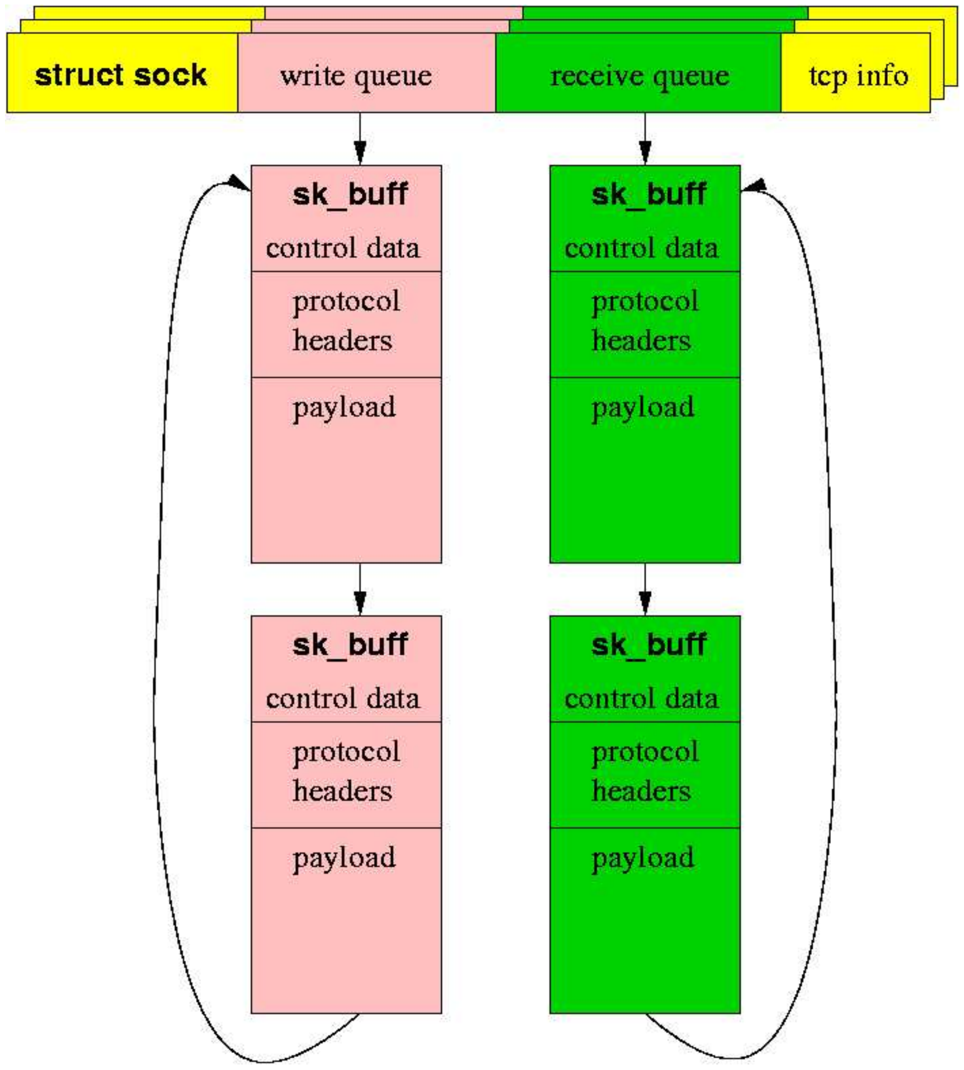
- Packets are stored in *Mbufs* (of 128 or 2048 bytes).
- TCP congestion control uses bytes in algorithms (e.g. initial cwnd = 1 * MSS = 1460)
- Little bookkeeping for each packet
- Better aligned with the IETF specifications (i.e.: many IETF activists have background with the BSD family)

Linux

- Packets are stored in *skbuffs* that are sized according to network interface MTU
- TCP congestion control uses packet counts (e.g. initial cwnd = 1)
- More bookkeeping for each packet (e.g. exact transmission time is stored for all packets)
- More “improvisation” on implementation

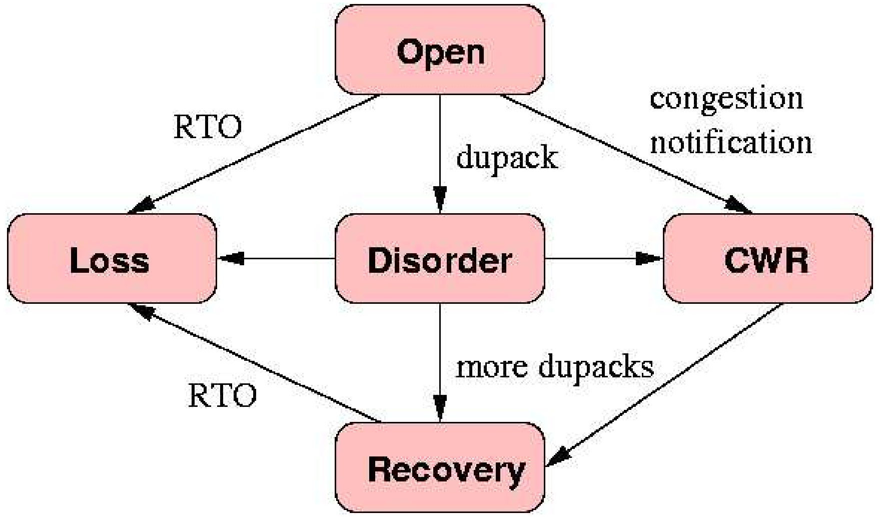
Data Structures

- Kernel-side correspondent for TCP socket is *struct sock*
- *struct sock* holds state data for the socket (such as the TCP variables regarding congestion window, etc.)
- There are several queue pointers
    - outgoing packets not yet acknowledged
    - incoming packets not yet delivered to application
- Queues hold chains of *sk_buffs*
    - *sk_buff* usually corresponds to one packet sent/received to network
    - In addition to packet data, there are protocol headers and control information
    - Note: instead of a single send/receive socket buffer there is just a chain of outgoing and incoming sk_buffs



Congestion control state machine

- Current state determines what to do with the congestion window
- ***Open***: The “fast path” of execution. Just transmit a new segment when valid ACK comes in and adjust the congestion window normally
- ***Disorder***: some data is unacknowledged, but it is considered to be reordering in the network for the present
- ***Recovery***: unacknowledged data is considered lost in the network. TCP sender should retransmit
- ***Loss***: RTO has expired. TCP sender should retransmit.
- ***CWR***: Other congestion notification than data loss has occurred (ECN, ICMP SQ, etc.)



Retransmission Engine

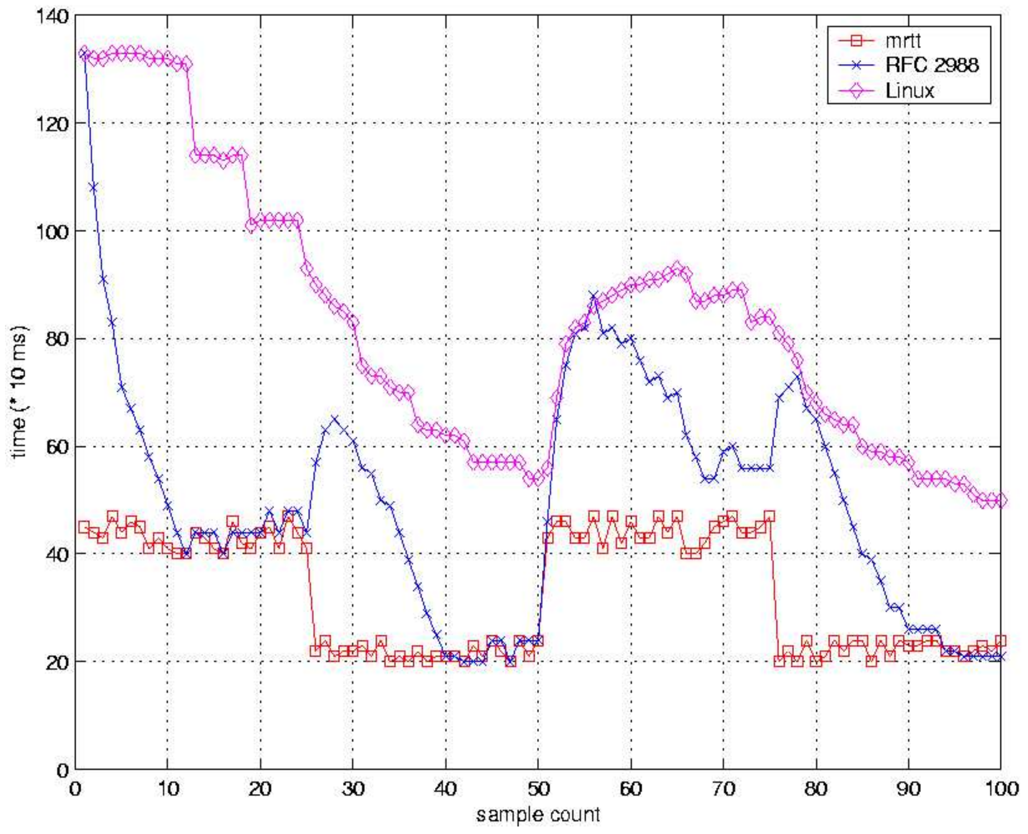
- Sender maintains the assumed state for each packet sent
  - Transmitted, acknowledged (by SACK), lost, retransmitted
  - Lost packet is always retransmitted ASAP (when allowed by cwnd), after which it is marked retransmitted
  - Recovery state: always mark the first unacknowledged packet lost (results in NewReno style retransmissions)
  - With SACK two alternatives:
    - FACK: all unacknowledged are marked lost when in Recovery state
    - conservative SACK: Consider possible packet reordering before marking packets lost
- Sender maintains estimate on number of packets in network (packets sent after cumack - pkts sacked - pkts lost + rexmits)
  - Above calculations are compared to congestion window and to determine when to transmit packet
- When RTO occurs, the sender marks all unacknowledged segments lost (although this is not always right decision)

Features (summary)

- RTO calculation (different to RFC 2988, see next slide)
- Explicit Congestion Notification (RFC 3168, no nonces)
- Detecting false retransmissions
  - With TCP Timestamps (similar to Eifel)
  - With DSACK enhancement using SACK option (RFC 2883)
- Limited Transmit (RFC 3042)
- Destination cache for storing TCP variables (like TCP Control Block Interdependence)
- Quick acknowledgements
- Congestion Window Validation

Linux RTO vs. RFC 2988

- Standard RTO calculation behaves weirdly with variable round-trip times
  - When RTT drops suddenly, RTO estimator gets higher value
  - When RTT is constant for a long while, small additional delay can trigger RTO
- Linux uses a minimum of 50 ms for RTT variance => RTO estimator does not get too near to RTT
- If RTT decreases, weight of variance term is reduced in the algorithm



To Kernel or Not To Kernel?

- Currently the TCP implementation is fixed undetachable part of kernel
- Sometimes it would be useful to have TCP as kernel module
  - Easier developement
  - Implementation would be easier to change
  - Switching implementation would be possible without rebooting
- One option would be to have TCP as a library in user-space
  - The congestion control algorithms would be carried out as a user process
  - Different user could have different flavors of TCP
  - Debugging would be very easy
  - However, number of difficult issues to be solved:
    - Performance
    - Security
    - Distribution of shared data (Path MTU, destination cache, etc.)
    - Policing (e.g. Ensuring that no one violates congestion control rules)

Summary

- Linux takes a rather different approach on implementing TCP than the traditional systems
- Linux “implicitly” includes quite a few of the TCP enhancements recently suggested in IETF and research papers
  - NewReno, two variants of SACK, DSACK, Eifel, ECN, Congestion Window Validation, Limited transmit, Rate-halving
  - It also includes features that are different from IETF specs
- Linux approach can also make some modifications difficult
  - A small modification somewhere can twist the packet markings and counters and cause surprising side-effects
- Unlike the earlier Linux versions, Linux 2.4 seems to be pretty well compliant to IETF specs and free from weird bugs