TCP: Implementation Details
Srinidhi Varadarajan

TCP Segment Size
- TCP segment size is not fixed. It is discovered during connection setup.
  - Set don’t fragment bit in IP header. If packet cannot be transmitted, router sends an ICMP error message
- Periodically TCP probes network to determine maximum segment size. Ideally TCP segment should fit within 1 IP fragment.
  - Fragmentation increases the number of packets.
  - Probability of loss increases since it is linked to number of packets not size of packets.

Window Updates
- Receiver returns available window size with each ack.
- After receiver advertises a wsize of 0, it readvertises a new non-zero window size after application retrieves some data from the TCP buffers
  - What happens if the window size update is lost?
- If sender doesn’t receive a windows size update for a long time after receiving a window size of 0, it can probe the receiver with a 1 byte segment to elicit a window update response

Nagle’s Algorithm
- Used to avoid excessive overhead for interactive TCP connections.
  - Problem: Telnet; a TCP packet is needed for each character typed. Ack are also needed for each character. ACK compression doesn’t really help
- Nagle’s algorithm: Intelligent buffering. If the transmission only involves one character, send the character out. Buffer characters till you get an ack and then send the buffered window.
  - For interactive applications, Nagle’s algorithm causes TCP to behave as a stop & wait protocol. This is not necessarily bad. Why?

Clark’s algorithm: Silly window syndrome
- Happens when receiving application reads data 1 byte at a time.
  - Problem: If receiving buffer is full TCP sends window updates for each byte of available buffer space.
- Solution is simple
  - Don’t send window updates till at least MSS bytes are available at the receiving buffer
- Nagle’s and Clark’s algorithms are complementary

Karn’s Algorithm and Timers
- When packets are retransmitted, acks can be ambiguous.
- Should you associate the ACK with the original transmission or the retransmission?
  - Associating the ACK with the original transmission can lead to longer estimates for RTT. The RTT estimate can eventually grow arbitrarily large
  - Associating the ACK with the retransmitted packet can lead to shorter estimates. For example, if network delay increases sharply, it will cause retransmissions. Since the ACKS were not lost, associating the ACKS with the retransmitted packet will lead to shorter RTT estimates
Karn’s Algorithm and Timers

- Simple solution: Ignore the RTT sample from a retransmitted packet.
  - Not so fast: What happens if the network delay increases sharply? The first packet that sees the higher delay will have a smaller timeout causing retransmission. If you ignore the RTT sample, RTT will never increase.
- Solution: Exponential timer backoff
  - Multiply timeout by a backoff factor, usually 2
  - Continue backing off till you get an ack for original transmission. Usually truncated backoff
  - Packets sent during timer backoff phase use backoff timer
  - All packets sent after end of timer backoff phase use correct RTT.

Fast Retransmit/Fast Recovery

- Fast Retransmit: On receiving 3 duplicate ACKS, retransmit packet with seq number just after ACK.
- Fast Recovery: Don’t reduce cwnd to 1.
  - ssthresh = cwnd/2
  - cwnd *= 3;
  - For each duplicate ACK after this
    - cwnd++;
  - When a non-duplicate ACK arrives
    - cwnd = ssthresh;
- Justification for fast recovery
  - If ACKS are arriving, some data must be getting through. The congestion was temporary. Otherwise, a packet would have timed out.