TCP Basic Design
- Port – mailboxes on machines
  o [www.iana.org](http://www.iana.org) (port assignment)
  o reserved ports < 1024
- Packet
  o To-port
  o From-port
  o Checksum
    □ for TCP header + TCP data only, IP checksum is for IP header only
    □ Is it even the right algorithm to detect network problems?
    □ Can’t detect “bit-flips”
- API: 2-way byte stream
  o Clients / servers
  o Connect / listen
  o Read / write (or vice versa)
  o From application’s perspective, it should be able to write/read arbitrary size of data, but does not need to be the same size on either sides (i.e. client or server)
  o Send / receive buffers
  o Data are divided into segments to be sent to the receiver
  o Receiver’s application will retrieve data from the receiver’s buffer
  o How to pick TCP segment size? IP MTU size – TCP header size
- Packet loss / ARQ
  o Timeouts and retransmissions
  o If checksum detects problem for the packet, host will not send an ACK
  o IP only hands to TCP the entire TCP packet after all fragments are ensembled
  o Need some unique number mechanism to distinguish retransmission packets (from sender and receiver ends)
- SCTP: transport protocol based on object model
- Sliding Window
  o To increase throughput of pipe
  o Propagation delay, transmission delay, bandwidth
  o Allow multiple packets to be send
  o Sliding window size? Depends on RTT * bandwidth of pipe.
    □ Has to be smaller than both the send and receive window sizes
  o Identify each packet by a sequence number
  o Also to need to identify ACK numbers
  o Label sequence # for packet or bytes? TCP: bytes
    □ Packet sequence #
      • Problem: if MTU estimation has changed in between retransmission, do we still retransmit the same amount of data?
Why do we have separate ACK #?
• Performance improvement
• Ability to combine sending data with ACKs
• Piggyback data along with ACK (e.g. GET request, ACK for the GET request as well as reply data)
• Delay ACKs
  o waits for 200ms to see if there are data to reply along with ACKs; if timeouts, just send back ACKs
  o design and implementation are much implemented
• Negative ACK: sent by receiver when it does not receive an expected packet (i.e. out-of-order packets)

- Fast retransmit
- Timeouts
  o Start with a fixed number (3 seconds)
  o Measure (exponentially weighted)
  o RTT – est = RTT-old * _ + RTT-new (1 – _)
  o Problems:
    ð What if loss? Timeout, then double length of timeout
    ð Congestion – increasing RTT
      • RTT – variance
      • Total timeout = RTT-est + variance * constant (e.g. 4)
- Flow control
  o If receiver is slow, sender is not allowed to send more packets than what the receiver can handle
  o Receive sends back (advertises) remaining receive window size back to sender
  o Once receiver buffer is full, sender will periodically asks receiver for the new receive window size
  o Silly window syndrome avoidance
    ð Only send back new receive window size if it is more than half full
  o Nagle’s algorithm:
    ð The slower the connection (or longer latency), the more bytes should be packaged up (versus sending one byte at a time) while transmitting data for applications like Telnet.