1. What are the functionalities of Transport Protocol?

- provide logical communication between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, messages to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP

2. What are the functionalities of UDP and TCP?

- relaiable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frillsextension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees

3. How does Multiplexing/Demultiplexing work in TCP/UDP?

- Demultiplexing at rcv host: delivering received segments to correct socket
- Multiplexing at send host: gathering data from multiple sockets, enveloping(包住) data with header (later used for demultiplexing)

4. How does Checksum work in UDP?

Sender

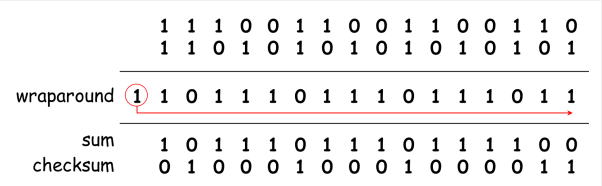
- treat sefment contents as sequence of 16-bit integers
- checksum: addition(1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver

- compute checksum of received segment

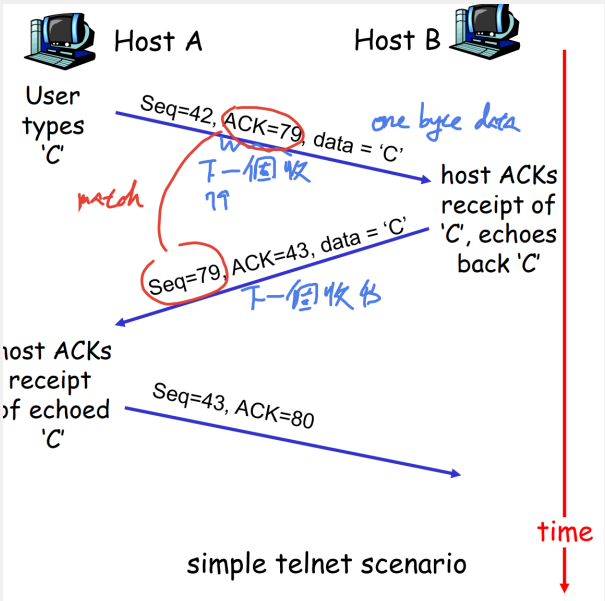
- check if computed checksum equals checksum field value
  - No - error detected
  - Yes - no error detected. But maybe errors nonetheless? More later...

    When adding numbers, a carryout from the most significant bit needs to be added to the result



5. What are sequence number and acknowledgement number in TCP? How does the sender and reciever use them?

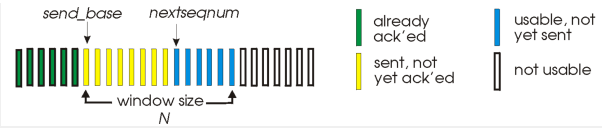
- sequence number:
  - byte stream "number" of first byte in segment's data
- acknowledgement number:
  - sequence number of next byte expected from other side
  - cumulative ACK



6. How do GO Back N and Selective Repeat work in TCP? What are the differences between them?

GO Back N

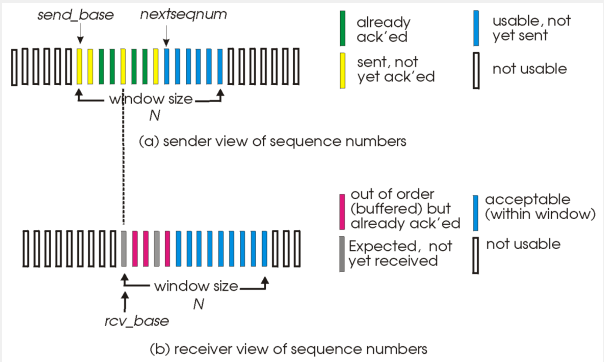
- k-bit sequece number in packet header
- "window" of up to N, consecutive unack'ed packets allowed



- ACK(n): ACKs all packets up to, including sequence number - "cumulative ACK"
- may deceive duplicate ACKs (see receiver)
- timer for each in-flight packet
- timeout(n): retransmit packet n and all higher sequence number packets in window
- ACK-only: always send ACK for correctly-received packet with highest in-order sequence number
  - may generate duplicate ACKs
  - need only remember **expectedseqnum**
- out-of-order packet:
  - discard (don't buffer) -> no receiver buffering
  - Re-ACK packet with highest in-order-seq #

Selective repeat

- receiver individually acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- sender window
  - N consecutive seq #'s
  - again limits seq #s of sent, unACKed pkts



7. How do Fast Retransmit and Fast Recovery work in TCP? What are the differences between them?

Fast Retransmit

- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs
  - Sender often sends many segments back-to-back

    - If segment is lost, there will likely be many duplicate ACKs
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
    - fast retransmit: resend segment before timer expires

Fast Recovery

- After resend the lost segment, TCP will enter fast recovery phase, until sender receive not duplicate ACK

8. How does Receive window work in TCP for the sender and receiver?

- The receive window is used to give the sender an idea about how much free buffer space is available at the receiver
- receiver: explicitly informs sender of (dynamically changing) amount of free buffer space
    - RcvWindow **field** in TCP segment
- sender: keeps the amount of transmitted, unACKed data less than most recently received RcvWindow: unACKed < RcvWindow

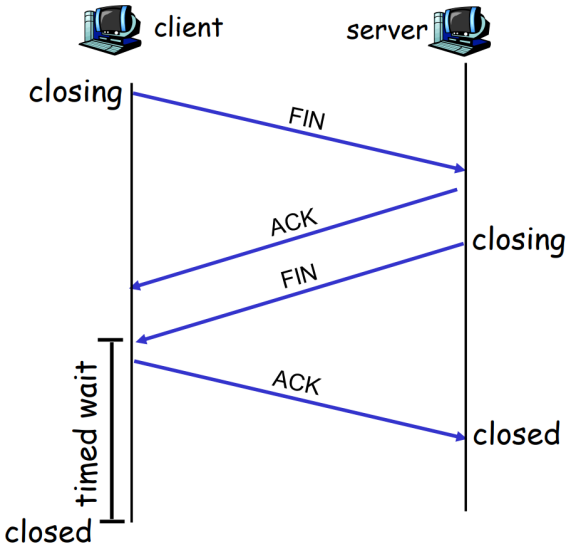
9. How does TCP establish and close a connection?

Three way handshake:

1. client host sends tCP SYN segment to server
    - specifies initial sequencial number
    - no data
2. server host receives SYN, replies with SYNACK segment
    - server allocates buffers
    - specifies server initial seq #
3. client receives SYNACK, replies with ACK segment, which may contain data

Closing a connection

1. client end system, sends TCP FIN control segment to server
2. server receives FIN, replies with ACK. Closes connection, sends FIN.
3. client receives FIN, replies with ACK.
    - Enters "timed wait" - will respond with ACK to received FINs
4. server, receives ACK. Connection closed

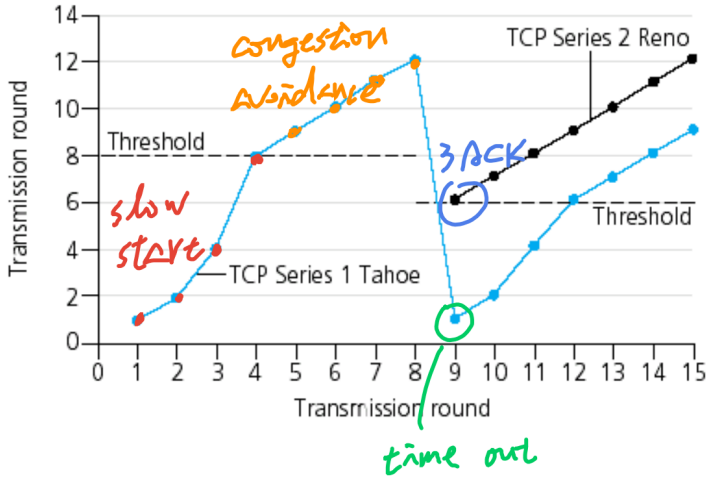


10.How does Slow Start work in TCP? Please describe the two phases and retransmission events with the two major versions

- When connection begins, CongWin = 1 MSS
  - Example: MSS = 500, bytes & RTT = 200 msec
  - initial rate = 20kbps
- available bandwidth maybe >> MSS/RTT
  - desirable to quickly ramp up to respectable rate
- When connection begins, increase rate exponentially fast until first loss event
- Summary: initial rate is slow but ramps up exponentially fast

1. When CongWin is below Threshold, sender in slow-start phase, window grows exponentially
2. When CongWin is above Threshold, sender is in congestion-avoidance phase, window grows linearly

- When a **triple duplicate ACK** occurs, Threshold set to CongWin/2 and ConWin set to Threshold
- When **timeout** occurs, Threshold set to CongWin/2 and CongWin is set to 1 MSS


Transmission round
Transmission round
Threshold
Threshold
TCP Series 1 Tahoe
TCP Series 2 Reno
congestion avoidance
slow start
3 ACK
time out