Transport Layer (III)

TCP (Transmission Control Protocol)

- Design parameters and objectives
  - used by most popular applications, majority of Internet traffic is transported over TCP
  - significant impact on congestion behavior of the Internet
  - must operate over networks with widely-varying characteristics
  - must be robust and (relatively) simple to implement

- TCP Header
  - Typically **20** bytes. More than 20 byte (up to 60 bytes) if the Options field is used.
  - **Sequence number** and **Acknowledgment number**: reliable data transfer service
  - **Window size** (receive window): flow control
  - **Data offset** (header length): 4-bit, specifies the length of the TCP header
  - **Options**: optional, variable-length, used when a sender and receiver negotiate the maximum segment size (MSS) or as a window scaling factor for use in high-speed networks.

TCP Header

Offsets Octet		0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port														
4	32	Sequence number																														
8	64	Acknowledgment number (if ACK set)																														
12	96	Data offset			Reserved 0 0 0					NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size													
16	128	Checksum																Urgent pointer (if URG set)														
20 ...	160 ...	Options (if Data Offset > 5, padded at the end with "0" bytes if necessary) ...																														

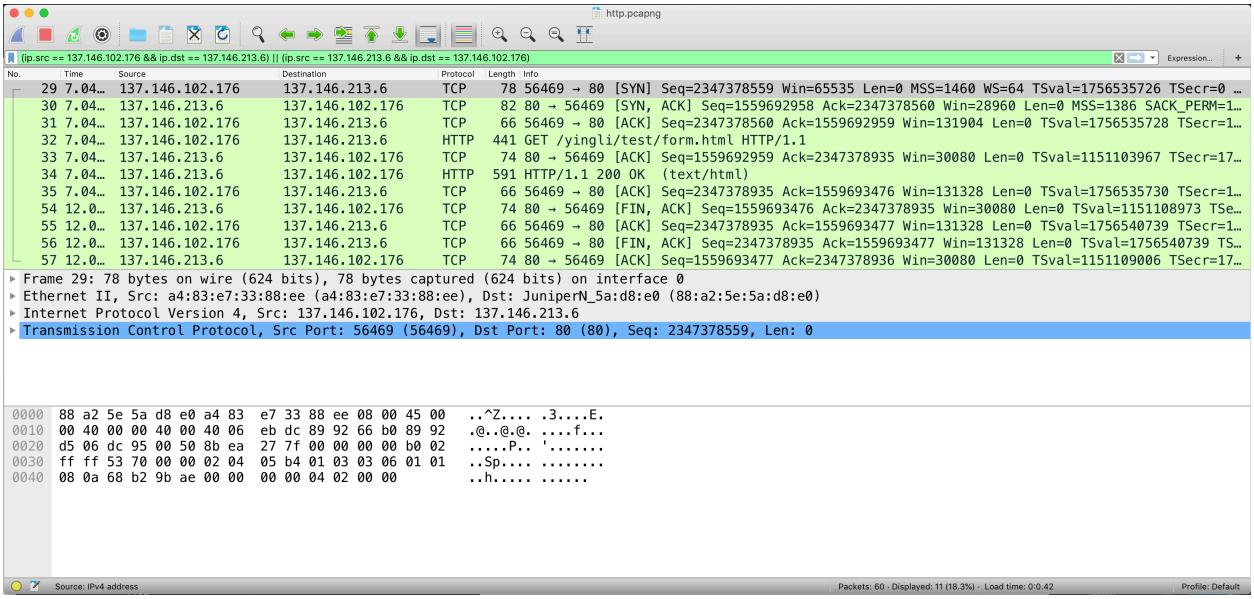
  - MSS is used by TCP to specify the largest amount of data a computer or communication device can receive in a single TCP segment.
  - MTU (Maximum Transmission Unit) is used by the IP layer to specify the largest packet size that can be sent over this path without suffering fragmentation.
- Flag fields:
  - **ACK**: indicate that the value carried in the acknowledgement field is valid; that is, the segment contains an acknowledgement for a segment that has been successfully received.
  - **RST, SYN**, and **FIN**: for connection setup and teardown
  - *CWR* and *ECE*: used in explicit congestion notification

  - **PSH**: indicate that the receiver should pass the data to the upper layer immediately
  - URG: indicate that there is data in this segment that the sending-size upper-layer entity has marked as “urgent”.

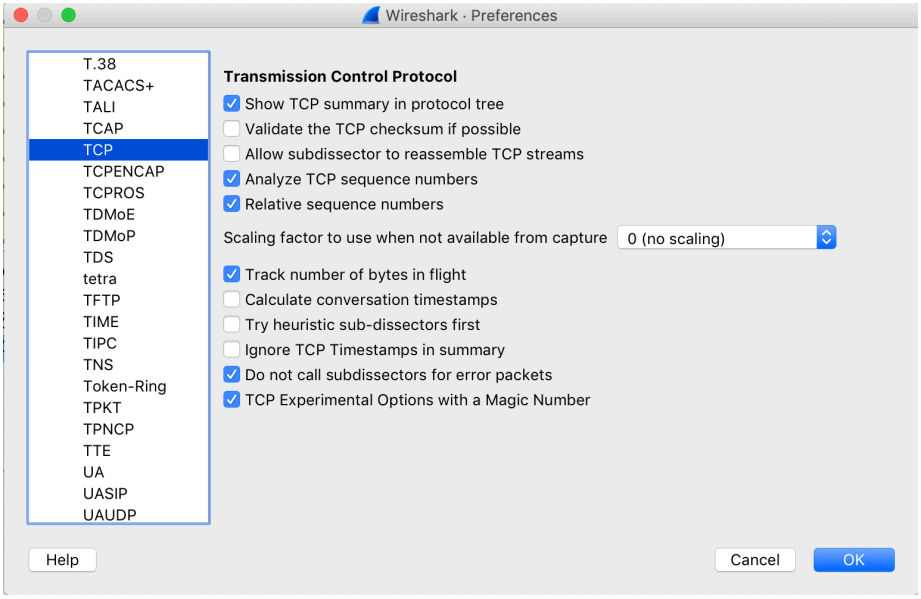
- How does TCP establish and terminate a connection? How does TCP leverage those header fields to provide reliable transmission (E.g., sequence number, acknowledgment number, etc)? Let’s use an example to illustrate.

- The following `http.pcapng` was a trace captured when opening http://cs.colby.edu/yingli/test/form.html. Filter: (ip.src == 137.146.102.176 && ip.dst == 137.146.213.6) || (ip.src == 137.146.213.6 && ip.dst == 137.146.102.176)

- The initial sequence/ACK numbers (2347378559/2347378560) are absolute sequence numbers randomly picked from 0 ~ 2^{32} (wrapped around) to avoid duplication.



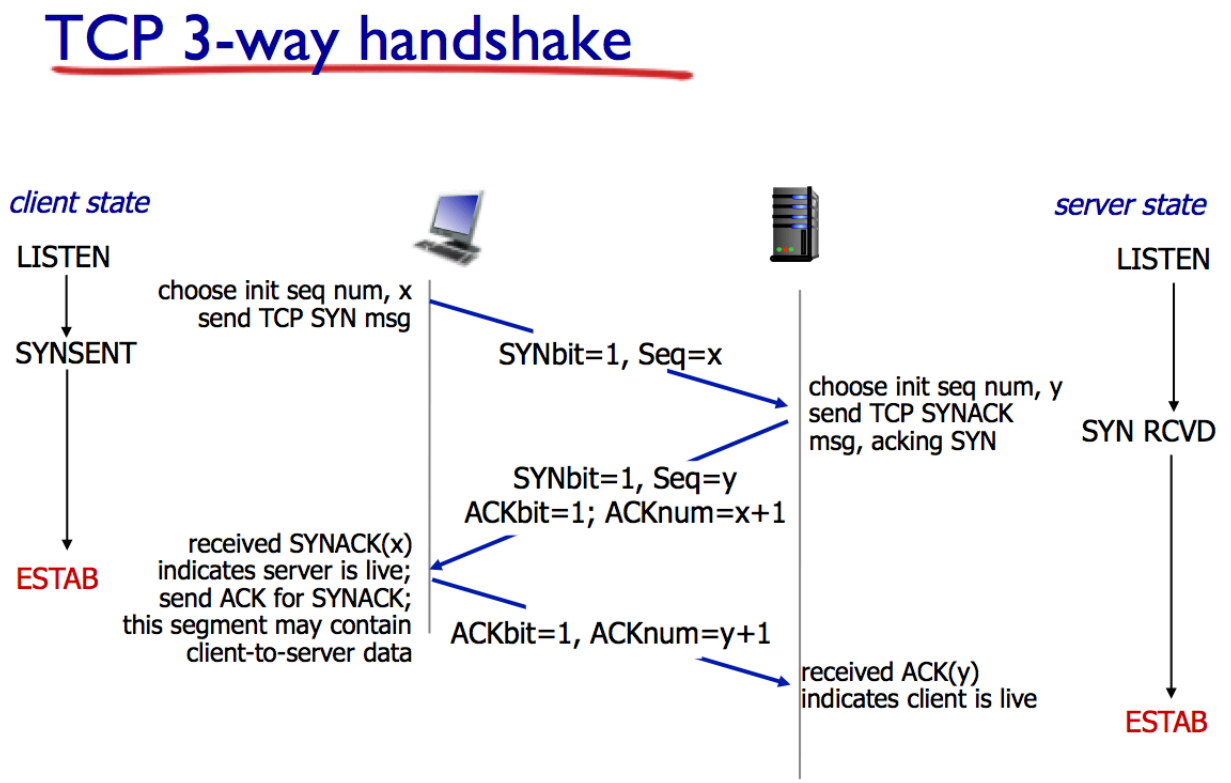
- Go to “Preference”, find “TCP” under “Protocols,” and check the option “Relative sequence numbers.” Then, Wireshark can display relative sequence numbers instead of absolute sequence numbers (shown in the screenshot above).



- As shown in the screenshot below, the sequence number of packet 29 becomes 0, and the ACK number of packet 30 becomes 1.

29	7.04…	137.146.102.176	137.146.213.6	TCP	78 56469 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1756535726 TSecr=0 SACK_PERM…
30	7.04…	137.146.213.6	137.146.102.176	TCP	82 80 → 56469 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1386 SACK_PERM=1 TSval=1151103965 …
31	7.04…	137.146.102.176	137.146.213.6	TCP	66 56469 → 80 [ACK] Seq=1 Ack=1 Win=131904 Len=0 TSval=1756535728 TSecr=1151103965

- To build a connection, TCP uses 3-way handshake. as shown in the screenshot above.



- To start, the client sends a packet with SYN field in the header set to 1 and Sequence Number set to X.
- After received the initial packet, the server sends an acknowledge packet with SYN set to 1, ACK field set to 1, Sequence Number set to Y, and ACK Number set to X + 1.
- Once the client received the response packet, it sends an acknowledgement packet to the server, with SYN set to 0, ACK set to 1, Sequence number set to X + 1, and ACK number

- After the TCP connection is set up, both ends can transmit data (shown in the screenshot below).

32	7.04…	137.146.102.176	137.146.213.6	HTTP	441 GET /yingli/test/form.html HTTP/1.1
33	7.04…	137.146.213.6	137.146.102.176	TCP	74 80 → 56469 [ACK] Seq=1 Ack=376 Win=30080 Len=0 TSval=1151103967 TSecr=1756535728
34	7.04…	137.146.213.6	137.146.102.176	HTTP	591 HTTP/1.1 200 OK (text/html)
35	7.04…	137.146.102.176	137.146.213.6	TCP	66 56469 → 80 [ACK] Seq=376 Ack=518 Win=131328 Len=0 TSval=1756535730 TSecr=1151103968

- The client sends a HTTP GET request to the server.

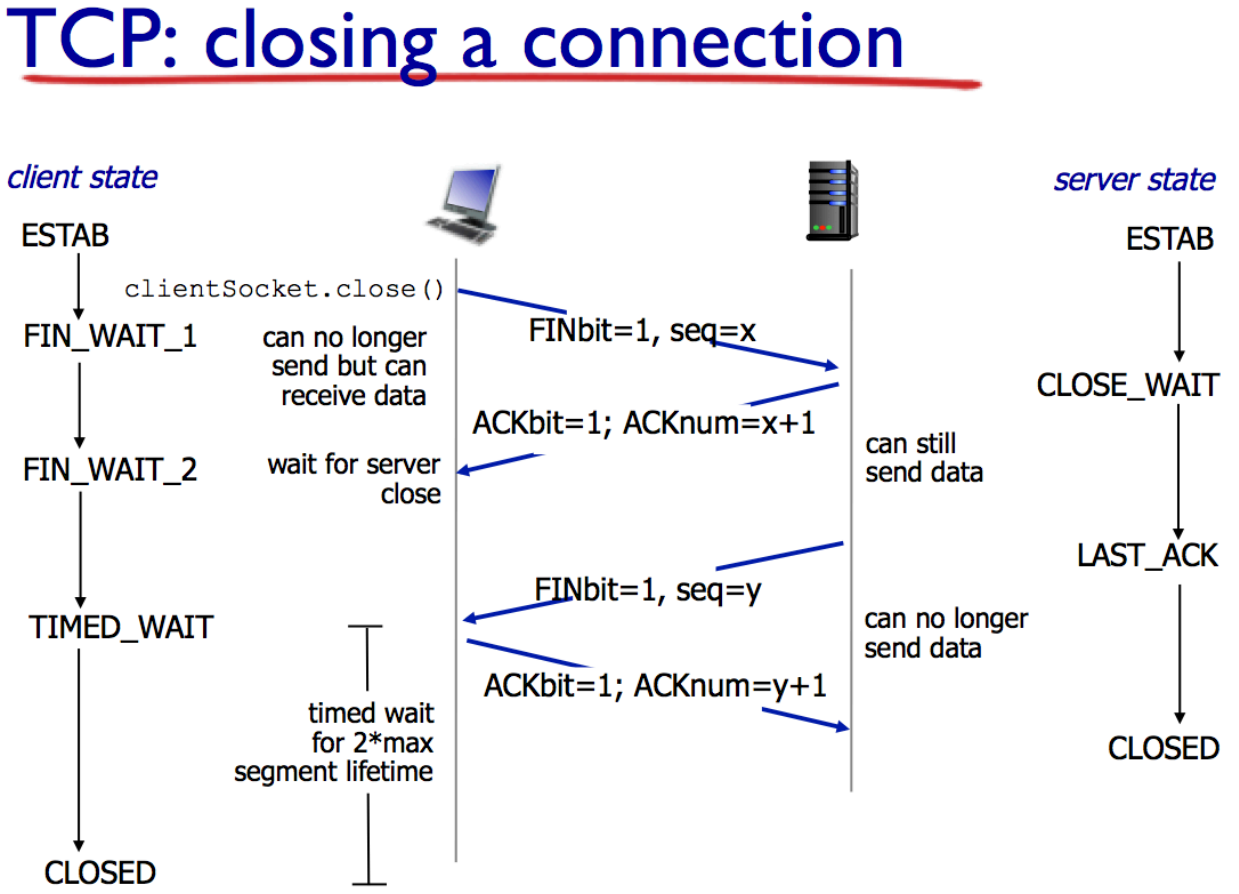
- The server sends an acknowledgement to the client, followed by a response to the GET request that includes the file the client request.
- The client sends an acknowledgment to the server after received the response.

- The next step is to close the TCP connection (shown in the screenshot below).

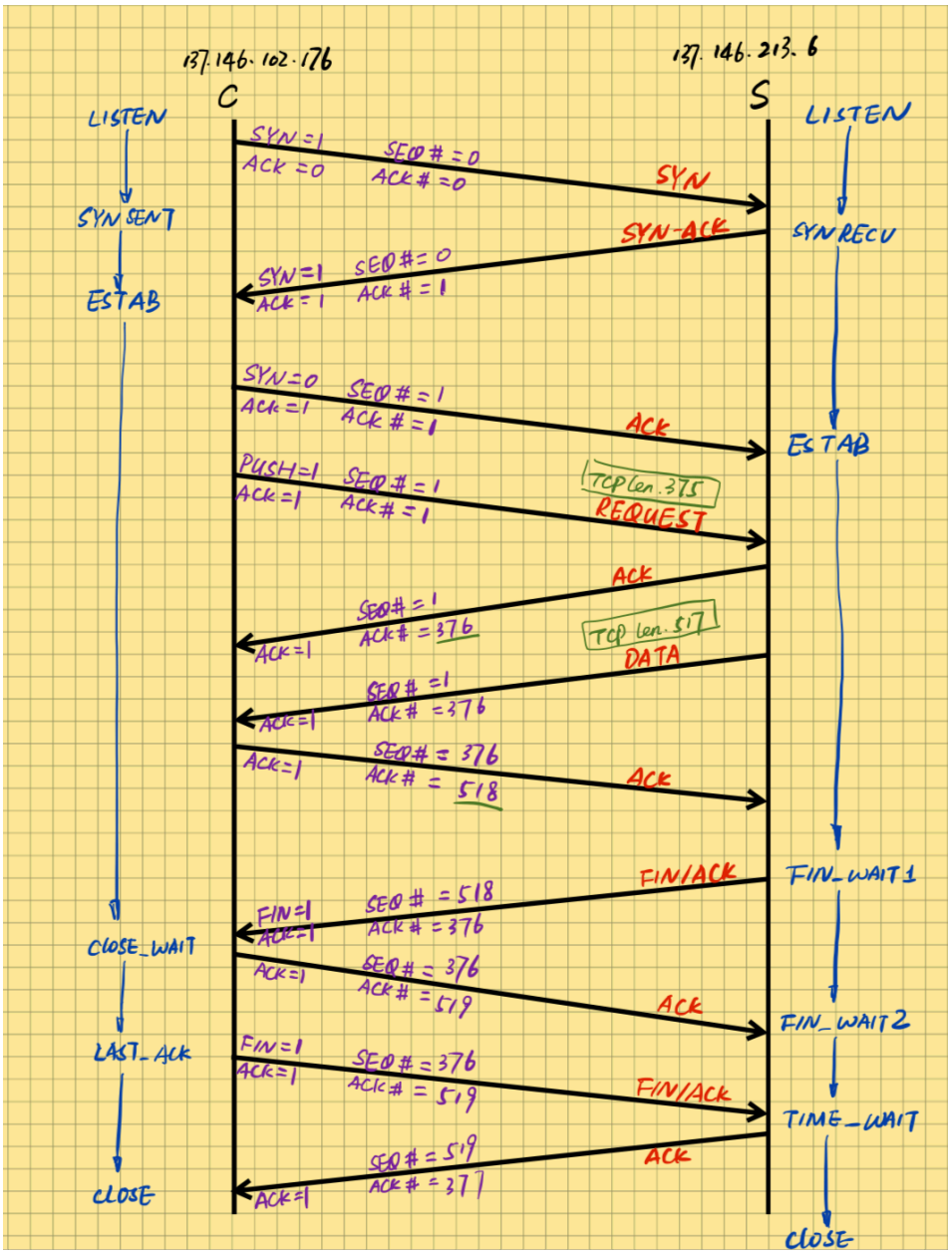
54	12.0…	137.146.213.6	137.146.102.176	TCP	74 80 → 56469 [FIN, ACK] Seq=518 Ack=376 Win=30080 Len=0 TSval=1151108973 TSecr=1756535730
55	12.0…	137.146.102.176	137.146.213.6	TCP	66 56469 → 80 [ACK] Seq=376 Ack=519 Win=131328 Len=0 TSval=1756540739 TSecr=1151108973
56	12.0…	137.146.102.176	137.146.213.6	TCP	66 56469 → 80 [FIN, ACK] Seq=376 Ack=519 Win=131328 Len=0 TSval=1756540739 TSecr=1151108973
57	12.0…	137.146.213.6	137.146.102.176	TCP	74 80 → 56469 [ACK] Seq=519 Ack=377 Win=30080 Len=0 TSval=1151109006 TSecr=1756540739

- To start the closing, either the server or the client can initialize the closing. In the above trace, the server sends a packet with FIN field in the header set to 1 and ACK set to 1.
- The client sends an acknowledge after received this close message with ACK set to 1.
- Then, the client also sends a close message to the server with FIN set to 1.
- The server sends an acknowledge packet to the server after received the server's close message.
- Once an end starts the closing process, it cannot send data anymore.

- The diagram gives a more detailed illustration including the state transition. Please note that the client initialize the closing in tis diagram.



- We can draw a sequence diagram for the HTTP trace shown above to better understand the process.

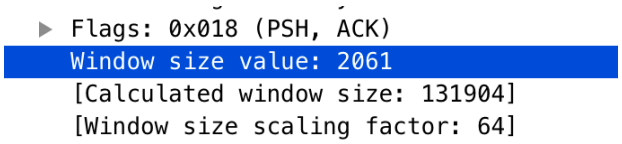


- Please note that TCP uses sequence and acknowledgement numbers to provide reliability.

TCP Congestion Control

- Receiver flow control
  - **Window Size field** - explicitly reported by the receiver
  - TCP Window Scale Option

- Calculated window size = Window size value x Window size scaling factor



  - Window size value: raw window size value read directly from TCP header
  - Calculated window size: calculated based on whether the scaling is allowed or not. If scaling is not allowed, scaling factor = 1. (The receiver window size may increment to calculated window size.)

- Network congestion control
  - **Retransmission timeout** - based on observed RTT and its variation
    - It's not a static value.
    - The timeout value will be exponentially increment after a timeout to alleviate the network congestion.
    - Why this makes sense?
      - The timer expiration is most likely caused by congestion in the network, that is, too many packets arriving at one/more router queues in the path between the source and destination, causing packets to be dropped and/or long queuing delays.
      - In times of congestion, if the sources continue to retransmit packets persistently, the congestion may get worse.
      - Instead, TCP acts more politely, with each seder retransmitting after longer a longer intervals.

  - **Transmission window** - based on detected packet loss
    - Source observes RTT and packet loss and adjusts transmission rate according to its estimate of the congestion state of the network
    - Transmission window provides a way to control the transmission rate

    - Transmission window size is proportional to the maximum transmission rate

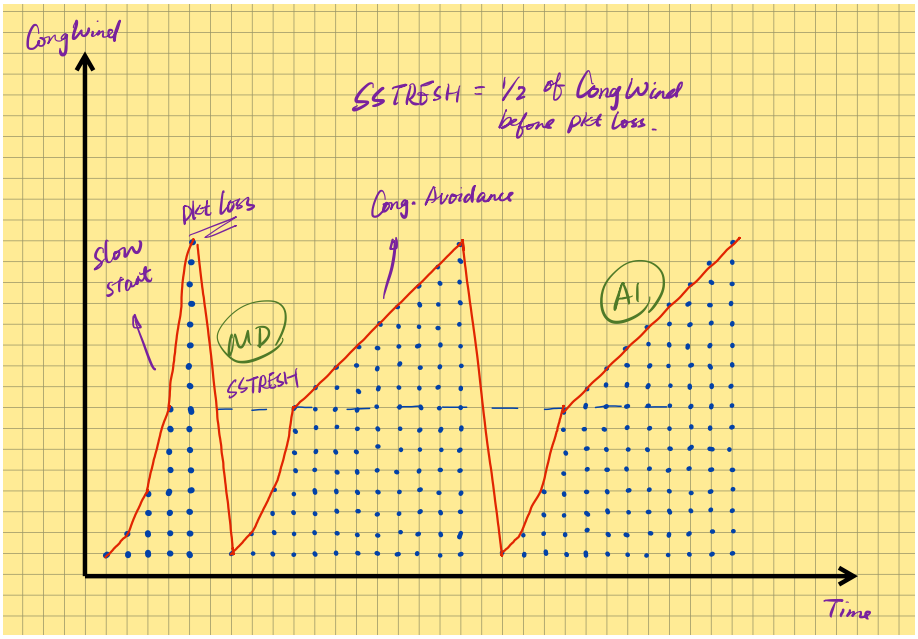
$$\text{TransWind} = \min(\text{RecvWind}, \text{CongWind})$$

      - TransWind - used in transmission
      - RecvWind - from Window Size field of the TCP header
      - CongWind - transmitter's estimate of how many unacknowledged packets can be pushed onto the network without causing congestion (maintained by the sender)
        - RFC 2001 defines the algorithm, Additive Increase Multiplicative Decrease (AIMD), to manage CongWind — better safe than sorry

- *Additive increase: increase CongWind by 1 message every RTT until loss detected*
- *Multiplicative decrease*: cut *CongWind* in half after loss

- Slow Start - initial growth of CongWind
  - Initially, the CongWind is 1 packet. After the receiver receives an acknowledgement, it increases the CongWind by 1 for each acknowledgement.
  - Once a packet loss is detected (timeout), the sender reset the CongWind to 1 and restart the slow start process.
  - The sender also set the **`ssthresh`** (slow start threshold) to CongWind/2, the value of CongWind is the value when the loss was last detected.
  - Once the CongWind reach at the `ssthresh`, it enters into Congestion Avoidance mode.

- Congestion Avoidance - AIMD-based “search” for optimal rate
  - In entry to the Congestion Avoidance mode, the congestion could be just around the corner!
  - TCP increase CongWind value by 1 packet every RTT.
  - Once the timeout occur (packet loss), Congestion Avoidance enters into Slow Start state.



- AIMD is used by TCP Reno.
- There are two other versions of TCP: CUBIC and BBR.
- Reno increases cautiously and backs off on loss; CUBIC uses smarter window growth; BBR estimates network capacity and adjusts its sending behavior.