Class Notes

Question 0:

What is maximum segement size (MSS)?

"The maximum amount of data that can be grabbed and placed in a segment is limited by the **maximum segment size (MSS).** The MSS is typically set by first determining the length of the largest link-layer frame that can be sent by the local sending host (the so-called **maximum transmission unit, MTU**), and then setting the MSS to ensure that a TCP segment (when encapsulated in an IP datagram) plus the TCP/IP header length (typically 40 bytes) will fit into a single link-layer frame. Both Ethernet and PPP link-layer protocols have an MTU of 1,500 bytes . Thus, a typical value of MSS is Approaches have also been proposed for discovering the path MTU—the largest link-layer frame that can be sent on all links from source to destination [RFC 1191]—and setting the MSS based on the path MTU value."

Source: Textbook

Question 1:

Is TCP like Go-BackN or Selective Repeat?

Answer:

TCP uses feature both from Go-BackN or Selective Repeat. For example:

(a) TCP uses cumulative acknowledgements (a feature in GoBackN)

(b) TCP can also use selectie acknowlegement as well (Proposed in RFC 2018). TCP receier can tell about selectie ACK using a special OPTION header in TCP header. So in this aspect it resembles Selective Repeat.

Question 2:

In the context of sender and receiver side windows, we established that as long as $W_s+W_r \leq 2^n$ condition holds, sender and receiver will not be confused and receiver will be able to reliably detect duplicates. But in the case of TCP, at least in theory, both sender and receiver windows can be of size 2^{32} -1 because there are 32 bit sequence no. In this case $W_s+W_r \leq 2n$ fails?

(Small note: Actually window size can not be 2^32 -1 in reality. See question 2 below after reading this answer.)

Answer:

If TCP were a simple protocol that only followed the rule , it would fail spectacularly. However, TCP uses a much more sophisticated mechanism to avoid this problem, which is based on **packet lifetime** rather than just the window size.

The simple rule is designed to prevent ambiguity in a scenario where a very old packet from a previous "lap" of the sequence numbers could be mistaken for a new packet. TCP solves this with two key mechanisms.

---

1. The Primary Solution: Maximum Segment Lifetime (MSL)

The core principle behind TCP's design is the **Maximum Segment Lifetime (MSL)**. This is a conservative estimate of the maximum time a packet can exist in the network before being discarded. The official standard sets it at **2 minutes**, though many modern systems use 30 or 60 seconds.

TCP's safety guarantee is based on this assumption: **The sequence numbers must not be able to be reused (or "wrap around") within the MSL.**

Let's do the math. The total sequence number space is 2^32 Bytes, which is about 4.3 GigaBytes.

- **Total Sequence Space:** 2^32 bytes
- **MSL (a safe, low value):** 60 seconds

To wrap the entire 4.3 GB sequence space in 60 seconds, you would need a sustained network speed of:

$$\text{Speed} = \frac{2^{32}\ \text{bytes}}{60\ \text{s}} \approx 71.5\ \text{MB/s}$$

Converting that to bits per second:

$$71.5\ \text{MB/s} \times 8\ \text{bits/byte} \approx 572\ \text{Mbps}$$

When TCP was designed in the 1970s and 80s, speeds of 572 Mbps were unimaginable. The assumption that sequence numbers wouldn't wrap within the MSL was incredibly safe. An old duplicate packet would long be expired and dropped by the network before its sequence number could ever be reused.

---

2. The Modern Solution: TCP Timestamps Option (PAWS)

Of course, today's multi-gigabit networks can easily exceed 572 Mbps. On these networks, it is possible to wrap the sequence numbers in just a few seconds, which is well within the MSL. This is where the **TCP Timestamps Option** comes in. This option, negotiated during the 3-way handshake, serves a crucial purpose called **PAWS (Protection Against Wrapped Sequence numbers)**.

- **How it works:** Each TCP segment includes a timestamp. The receiver keeps track of the latest timestamp it has seen. If a packet arrives that appears to be valid in the sequence number space but has a timestamp that is *older* than the last one seen, the receiver knows it's a stale packet from a previous "lap." The packet is identified as a duplicate and safely discarded.

The timestamp effectively extends the sequence number space, allowing the receiver to differentiate between an old segment and a new segment that happen to share the same sequence number.

Conclusion

So, to summarize:
You're right that the simple $Ws+Wr \leq 2^n$ rule doesn't hold for TCP. Instead of limiting the window size, TCP relies on:

1. **Maximum Segment Lifetime (MSL)** to ensure old packets die off on slower networks.
2. **TCP Timestamps (PAWS)** to reject old duplicate packets on modern, high-speed networks.

Question 3:

In TCP packet, size of receive windows is 16 bits. That means a receiver can advertise only 64K size of window. Is that enough in modern high-speed networks?

Answer:

A 16-bit receive window, which allows a maximum size of only 65,535 bytes (64 KiB), is **not enough** for modern high-speed networks.

A small window size can become a major bottleneck, especially on networks with high bandwidth and long latency (often called "long fat networks"). This is due to a concept called the **Bandwidth-Delay Product (BDP)**.

(Note: Here delay means round-trip time.)

BDP calculates the maximum amount of data that can be "in flight" on the network at any given time.

If the receive window is smaller than the BDP, the sender will have to stop and wait for an acknowledgment after sending just 64 KiB of data, even if the network path could hold much more. This prevents the sender from fully utilizing the available network capacity, leading to poor performance.

The Solution: TCP Window Scale Option

To solve this exact problem, **TCP Window Scale** was introduced as an option in the TCP header (defined in RFC 7323). This option allows the receiver to advertise a much larger receive window.

Here's how it works:

1. **Negotiation:** The window scale factor is negotiated during the initial TCP three-way handshake (in the SYN and SYN-ACK packets). Both sides must include the option for it to be used.
2. **Scale Factor:** The option specifies a **scale factor**, which is an 8-bit number. This number represents how many bits to left-shift the 16-bit window size value in the main TCP header. The scale factor can range from 0 to 14.
3. **Calculation:** The true receive window size is calculated as:

$$\text{True Window Size} = \text{Window Size from Header} \times 2^{\text{scale factor}}$$

Example

Let's say a receiver wants to advertise a receive window of approximately 1 megabyte (1,048,576 bytes).

- It can't put `1,048,576` in the 16-bit window field.
- Instead, it can set a **scale factor of 4** during the handshake (2^4 = 16).

- Then, it places the value `65,535` in the 16-bit window field.
- The sender then calculates the true window size: 65,535 * 2^4 = 1048,560 Bytes

With the maximum scale factor of 14, TCP can support a receive window of nearly 1 gigabyte (), which is more than sufficient for today's high-speed networks.

TCP connection state transition diagram

From RFC 793:

```
                              +---------+ ---------\      active OPEN
                              |  CLOSED |            \    -----------
                              +---------+<---------\   \   create TCB
                                |     ^              \   \  snd SYN
                   passive OPEN |     |   CLOSE        \   \
                   ------------ |     | ----------       \   \
                    create TCB  |     | delete TCB         \   \
                                V     |                      \   \
                              +---------+            CLOSE    |    \
                              |  LISTEN |          ---------- |     |
                              +---------+          delete TCB |     |
                   rcv SYN      |     |     SEND              |     |
                  -----------   |     |    -------            |     V
 +---------+      snd SYN,ACK  /       \   snd SYN          +---------+
 |         |<-----------------           ------------------>|         |
 |   SYN   |                    rcv SYN                     |   SYN   |
 |   RCVD  |<-----------------------------------------------|   SENT  |
 |         |                    snd ACK                     |         |
 |         |------------------           -------------------|         |
 +---------+   rcv ACK of SYN  \       /  rcv SYN,ACK       +---------+
   |           --------------   |     |   -----------
   |                  x         |     |     snd ACK
   |                            V     V
   |  CLOSE                   +---------+
   | -------                  |  ESTAB  |
   | snd FIN                  +---------+
   |                   CLOSE    |     |    rcv FIN
   V                  -------   |     |    -------
 +---------+          snd FIN  /       \   snd ACK          +---------+
 |  FIN    |<-----------------           ------------------>|  CLOSE  |
 | WAIT-1  |------------------                              |   WAIT  |
 +---------+          rcv FIN  \                            +---------+
   | rcv ACK of FIN   -------   |                            CLOSE  |
   | --------------   snd ACK   |                           ------- |
   V        x                   V                           snd FIN V
 +---------+                  +---------+                   +---------+
 |FINWAIT-2|                  | CLOSING |                   | LAST-ACK|
 +---------+                  +---------+                   +---------+
   |                rcv ACK of FIN |                 rcv ACK of FIN |
   |  rcv FIN       -------------- |    Timeout=2MSL -------------- |
   |  -------              x       V    ------------        x       V
    \ snd ACK                 +---------+delete TCB         +---------+
     ------------------------>|TIME WAIT|------------------>| CLOSED  |
                              +---------+                   +---------+
```

TCP Connection State Diagram
Figure 6.

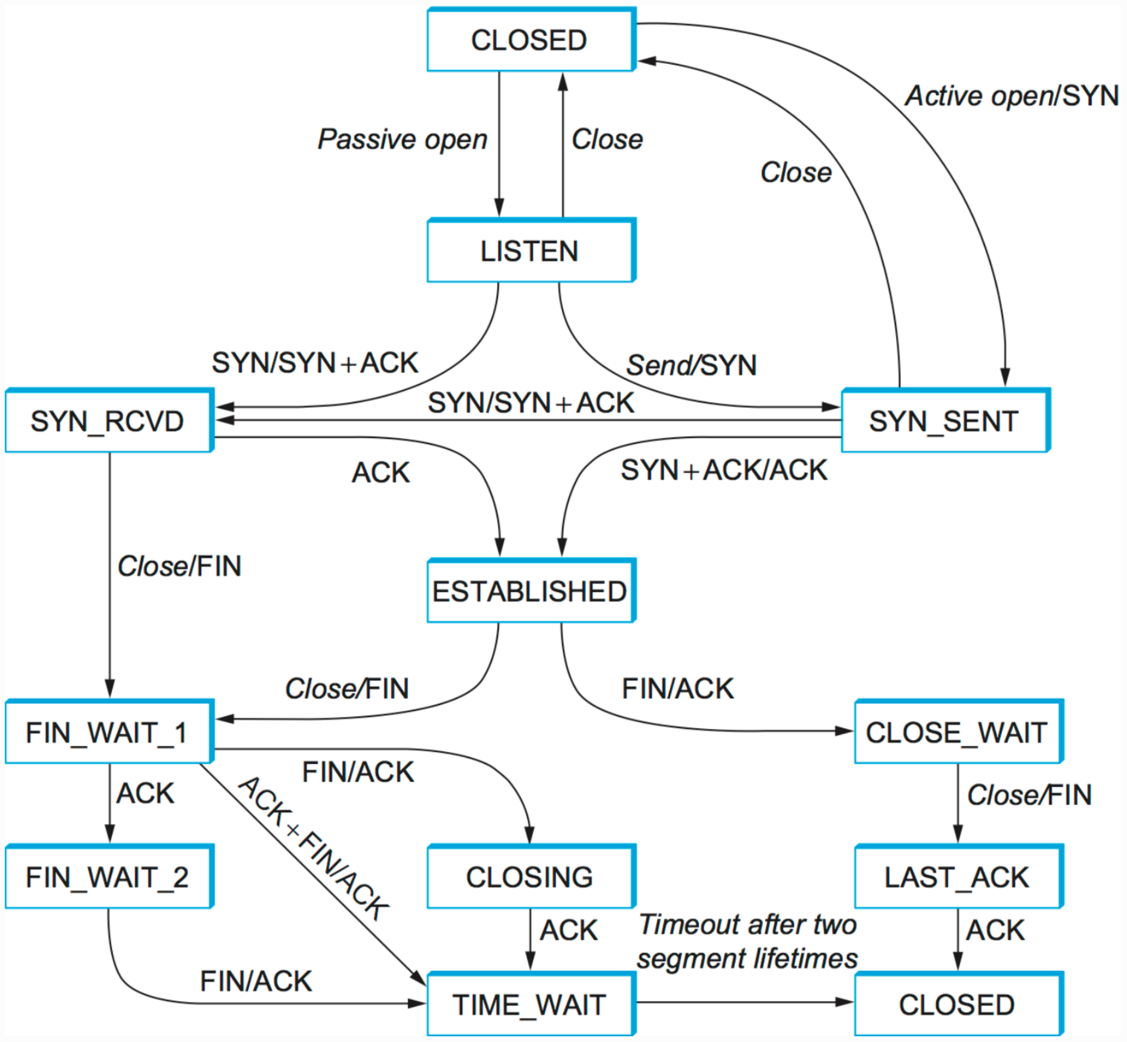
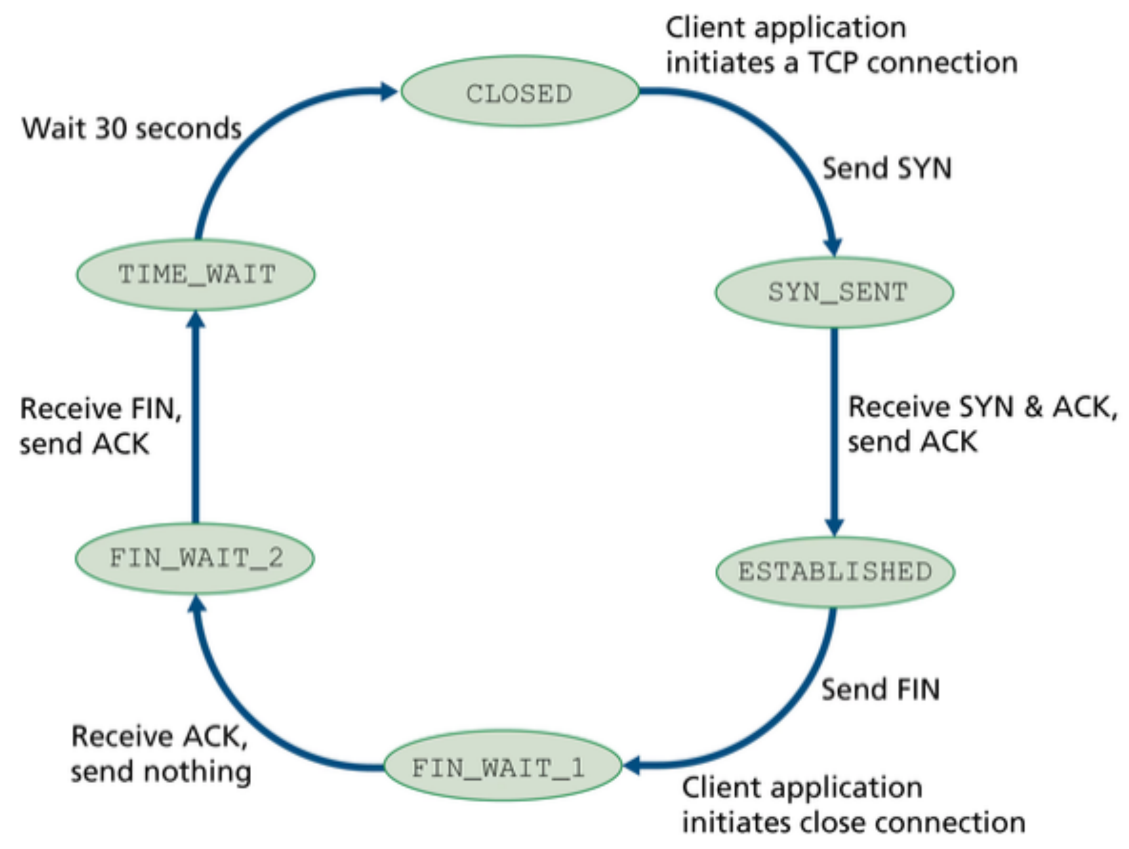


Figure 131. *TCP state-transition diagram.*

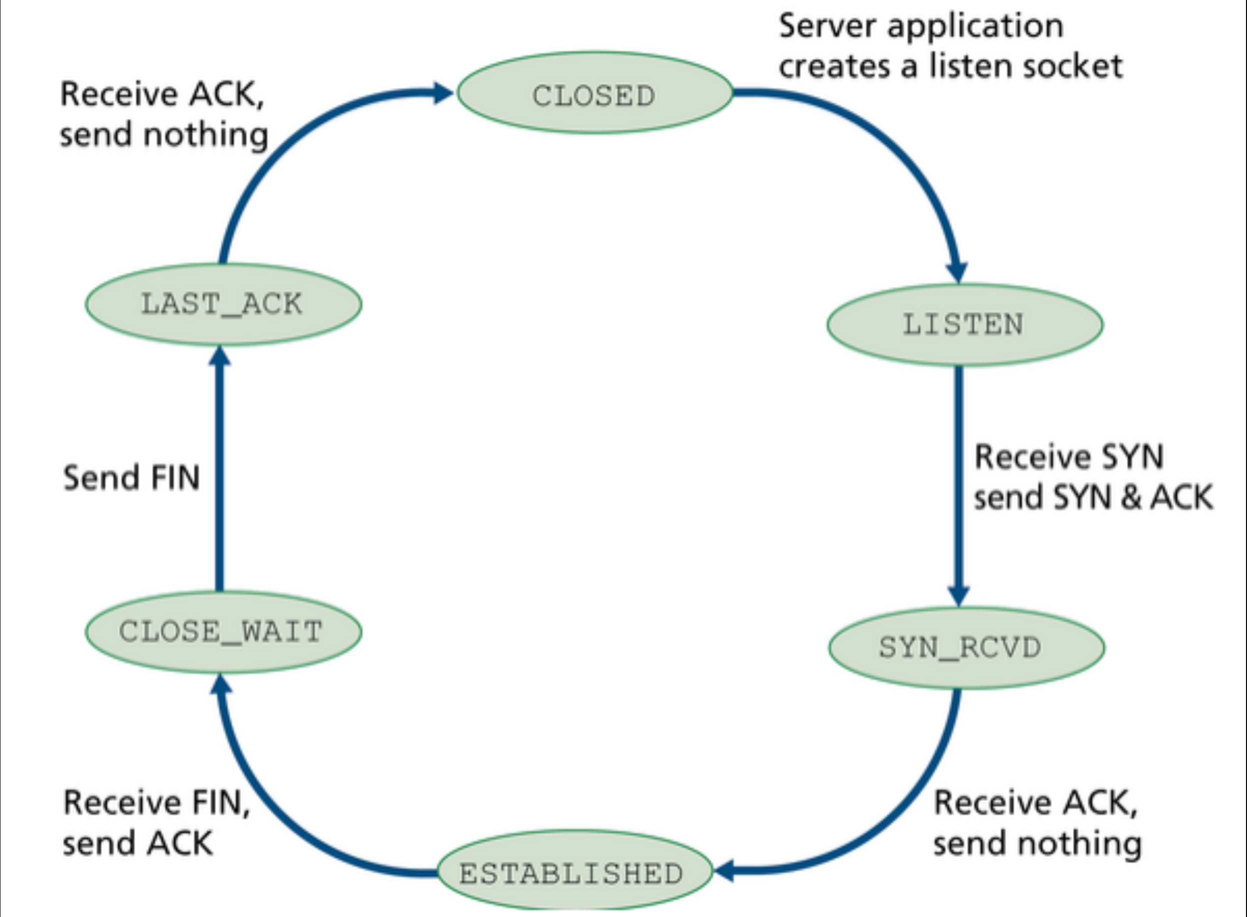
Source: https://book.systemsapproach.org/e2e/tcp.html

Client side:


Client application initiates a TCP connection
CLOSED
Wait 30 seconds
Send SYN
TIME_WAIT
SYN_SENT
Receive FIN, send ACK
Receive SYN & ACK, send ACK
FIN_WAIT_2
ESTABLISHED
Send FIN
Receive ACK, send nothing
FIN_WAIT_1
Client application initiates close connection

Source: TextBook

Server side:



Source: TextBook

- Client side closes first: ESTABLISHED → FIN_WAIT_1 → FIN_WAIT_2 → TIME_WAIT → CLOSED.
- The other side (source) closes first: ESTABLISHED → CLOSE_WAIT → LAST_ACK → CLOSED.
- (Exercise for students): Both sides close at the same time:
  ESTABLISHED → FIN_WAIT_1 → CLOSING → TIME_WAIT → CLOSED.

Wait in TIME_WAIT is about 2 minutes (maximum time an IP packet is believed to be alive in the IP network.)

The **TIME_WAIT** state in TCP is typically set to a duration that is **twice the Maximum Segment Lifetime (MSL)**, as per the TCP specification.

Typical Value:

- The **MSL** is usually assumed to be **60 seconds**, meaning the typical value of **TIME_WAIT** is:
  - **TIME_WAIT = 2 × MSL = 2 × 60 seconds = 120 seconds** (or 2 minutes).

See: RFC 793 for details.

See section 5.2.3 in https://book.systemsapproach.org/e2e/tcp.html for further discussion of TCP connection management.