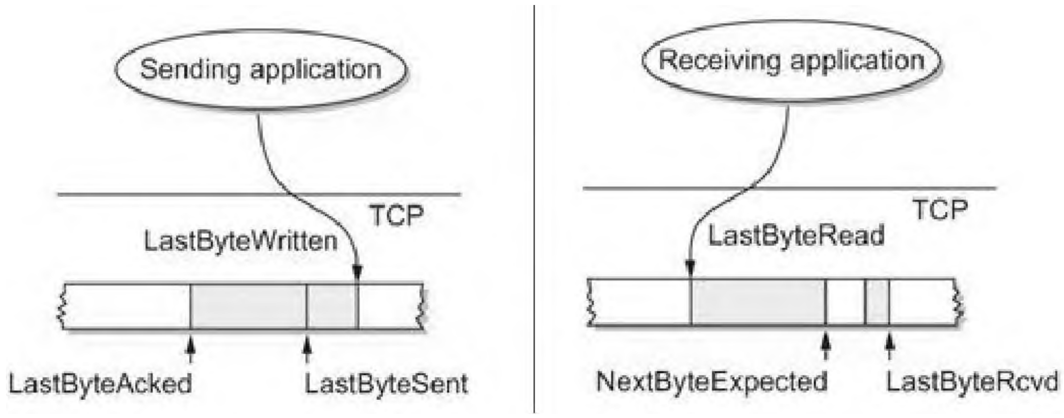
5.6 TCP FLOW CONTROL

- TCP uses a variant of sliding window known as adaptive flow control that:
    a. guarantees *reliable* delivery of data
    b. ensures *ordered* delivery of data
    c. enforces *flow control* at the sender
- Receiver advertises its window size to the sender using AdvertisedWindow field. Sender thus cannot have *unacknowledged* data greater than AdvertisedWindow.



Send Buffer

• Sending TCP maintains *send buffer* which contains 3 segments
(1) acknowledged data (2) unacknowledged data (3) data to be transmitted.
• Send buffer maintains three *pointers*
(1) LastByteAcked, (2) LastByteSent, and (3) LastByteWritten such that:
LastByteAcked* ≤ *LastByteSent* ≤ *LastByteWritten
• A byte can be sent only *after* being written and only a sent byte *can be* acknowledged.
• Bytes to the *left* of LastByteAcked are not kept as it had been acknowledged.

Receive Buffer

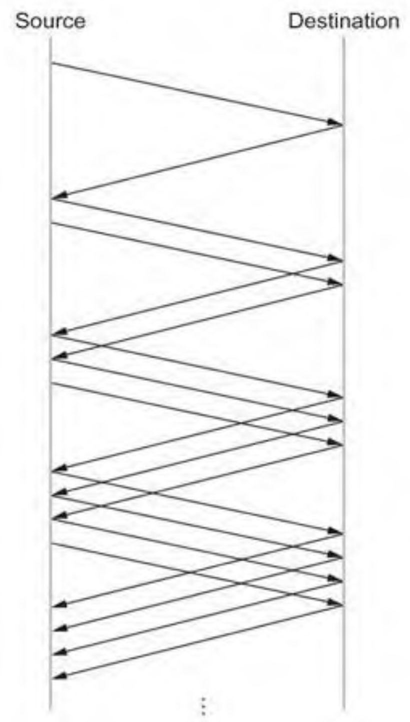
• Receiving TCP maintains *receive* buffer to hold data even if it arrives out-of-order.
• Receive buffer maintains three *pointers* namely (1) LastByteRead, (2) NextByteExpected, and (3) LastByteRcvd such that:
LastByteRead* ≤ *NextByteExpected* ≤ *LastByteRcvd + 1
• A byte *cannot* be read until that byte and all preceding bytes have been received.
• If data is received *in order*, then NextByteExpected = LastByteRcvd + 1
• Bytes to the *left* of LastByteRead are not buffered, since it is read by the application.

5.7 TCP CONGESTION CONTROL

- Congestion occurs if load (number of packets sent) is greater than capacity of the network (number of packets a network can handle).
- When load is less than network capacity, throughput increases proportionally.
- When load exceeds capacity, queues become full and the routers discard some packets and throughput declines sharply.
- When too many packets are contending for the same link
  1. The queue overflows
  2. Packets get dropped
  3. Network is congested
- Network should provide a congestion control mechanism to deal with such a situation.
- TCP maintains a variable called *CongestionWindow* for each *connection*.
- TCP Congestion Control mechanisms are:
  1. Additive Increase / Multiplicative Decrease (AIMD)
  2. Slow Start
  3. Fast Retransmit and Fast Recovery

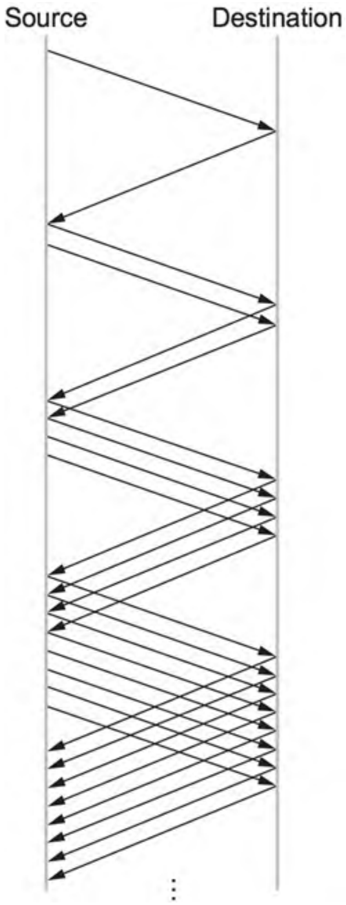
Additive Increase / Multiplicative Decrease (AIMD)

- TCP source *initializes* CongestionWindow based on congestion level in the network.
- Source *increases* CongestionWindow when level of congestion goes down and *decreases* the same when level of congestion goes up.
- TCP interprets *timeouts* as a sign of congestion and reduces the rate of transmission.
- On timeout, source reduces its CongestionWindow by half, i.e., *multiplicative decrease*. For example, if CongestionWindow = 16 packets, after timeout it is 8.
- Value of CongestionWindow is never less than maximum segment size (MSS).
- For *example*, when ACK arrives for 1 packet, 2 packets are sent. When ACK for both packets arrive, 3 packets are sent and so on.
- CongestionWindow increases and decreases throughout *lifetime* of the connection.



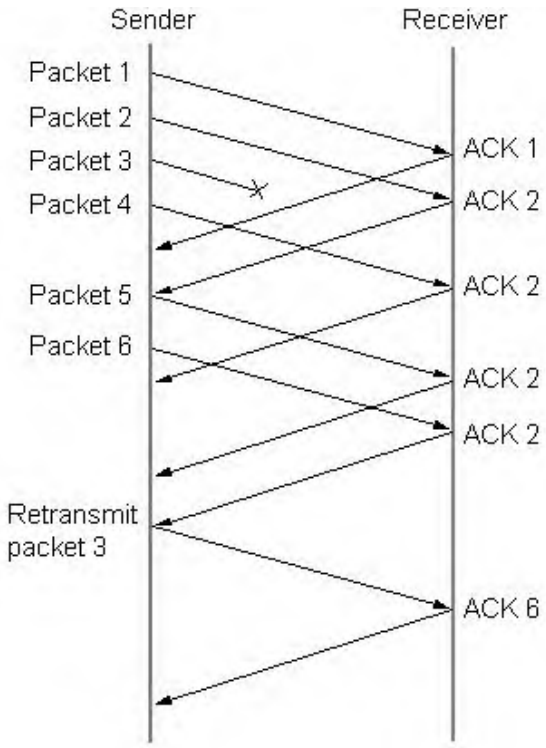
Slow Start

- Slow start is used to increase CongestionWindow *exponentially* from a cold start.
- Source TCP *initializes* CongestionWindow to one packet.
- TCP *doubles* the number of packets sent every RTT on successful transmission.
- When ACK arrives for first packet TCP adds 1 packet to CongestionWindow and sends two packets.
- When two ACKs arrive, TCP increments CongestionWindow by 2 packets and sends four packets and so on.
- Instead of sending entire permissible packets at once (bursty traffic), packets are sent in a phased manner, i.e., *slow start*.
- Slow start is repeated until CongestionWindow reaches CongestionThreshold and thereafter 1 packet per RTT.



Fast Retransmit And Fast Recovery

- TCP timeouts led to long periods of time during which the connection went dead while waiting for a timer to expire.
- Fast retransmit is a heuristic approach that *triggers* retransmission of a dropped packet sooner than the regular timeout mechanism. It *does not* replace regular timeouts.
- When a packet arrives out of order, receiving TCP resends the same acknowledgment (*duplicate ACK*) it sent last time.
- When *three duplicate* ACK arrives at the sender, it infers that corresponding packet may be lost due to congestion and retransmits that packet. This is called ***fast retransmit*** before regular timeout.
- When packet loss is detected using fast retransmit, the slow start phase is replaced by additive increase, multiplicative decrease method. This is known as ***fast recovery***.
- Instead of setting CongestionWindow to one packet, this method uses the ACKs that are still in pipe to clock the sending of packets.


Sender
Receiver
Packet 1
Packet 2
Packet 3
Packet 4
Packet 5
Packet 6
Retransmit packet 3
ACK 1
ACK 2
ACK 2
ACK 2
ACK 2
ACK 6