Project 3
Flow and congestion control

15-441 Spring 2010

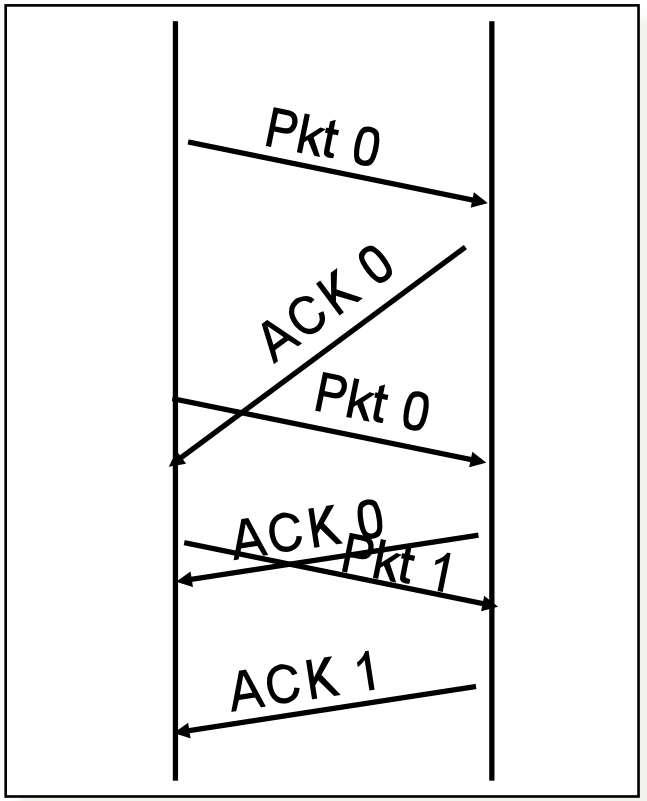
Recitation #10

Context – Project 3

- Bit torrent-like file transfer app
  - Use UDP to understand the challenges in transport protocol design and implementation
- Checkpoints
  - #1: Generate WHOHAS queries and IHAVE replies (due today!)
  - #2: Download a chunk using stop and wait
  - #3: Add flow control
  - #4: Add congestion control

Why do we need flow control

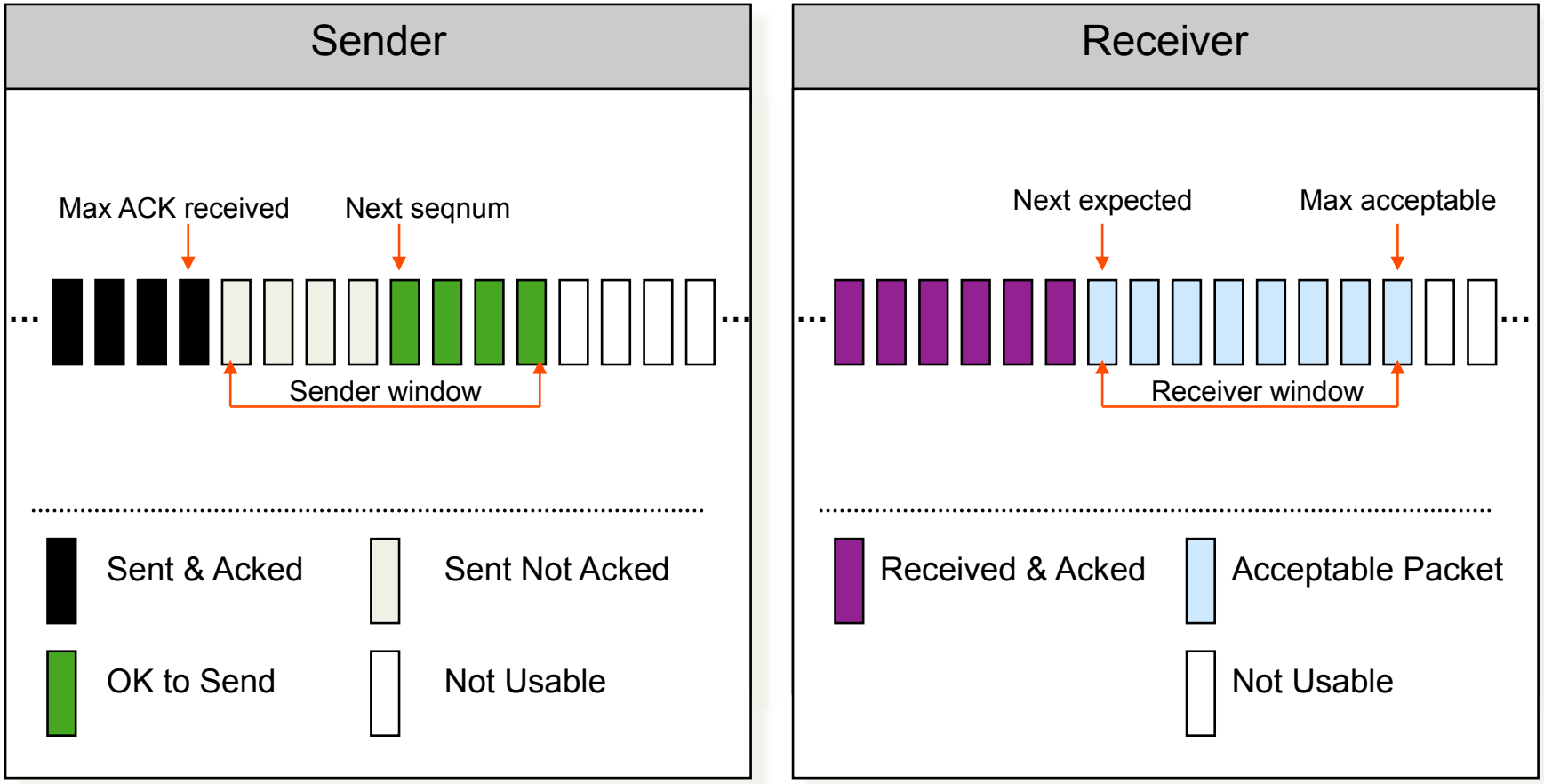
- Stop and wait
  - Don't transmit a new packet until you get an ACK
- Limited performance
  - 1 packet per RTT



Sliding window - idea

- Enough in-flight packets to “hide” the RTT
- Have a buffer of packets on each side
  - Advance window when sender and receiver agree packets at the beginning have been received
- Common case
  - Receiver: send cumulative ACK, slide window
  - Sender: slide window, send next packet

Sliding window - visualization



Sliding window – reliability

- Lost packet (with cumulative ACKs)
  - When detecting a missing packet, the receiver sends an ACK with the sequence number of the last correctly ordered packet
  - If the sender timeouts while waiting for an ACK, it will retransmit
- Accommodating out of order packets
  - The sender waits for 3 duplicate ACKs before retransmitting

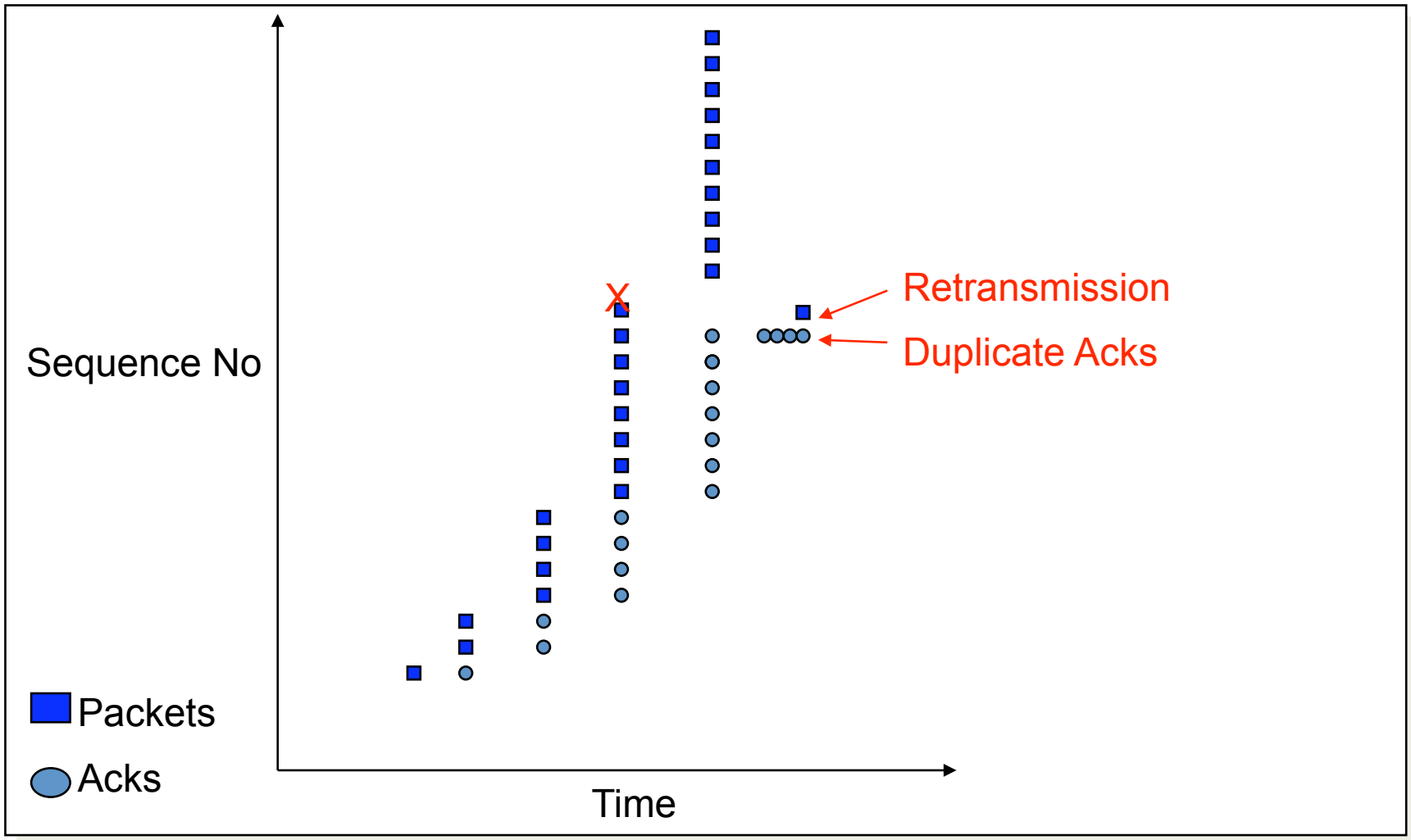
Sliding window – implementation

- Sender keeps 3 pointers
  - LastPacketAcked, LastPacketSent, LastPacketAvailable
- Ensure invariants
  - LastPacketAcked <= LastPacketSent
  - LastPacketSent <= LastPacketAvailable
  - LastPacketAvailable-LastPacketAcked <= WindowSize
- Simplify by ignoring receiver issues: he can always process whatever he gets

Sliding window – testing

- Use assert statements to ensure invariants
- Inject some losses to test loss behavior
- Write time-stamped packet sequence numbers and ACKs to a file and plot them

Sliding window – testing fast retrans



Congestion control

- Adapt to the network conditions by changing your window size
- Losses assumed to be due to congestion, so throttle back the sender when you see one
- Implement:
  - Slow start
  - Congestion Avoidance
  - Fast retransmit

Slow start

- Start with window of size 1 (SW=1) and define the slow start threshold to be 64 (ssthresh=64)
- Now, upon:
  - ACK received: SW++
  - Loss detected: ssthresh = max(SW/2,2), SW=1
  - SW = ssthresh: move into congestion avoidance

Congestion avoidance

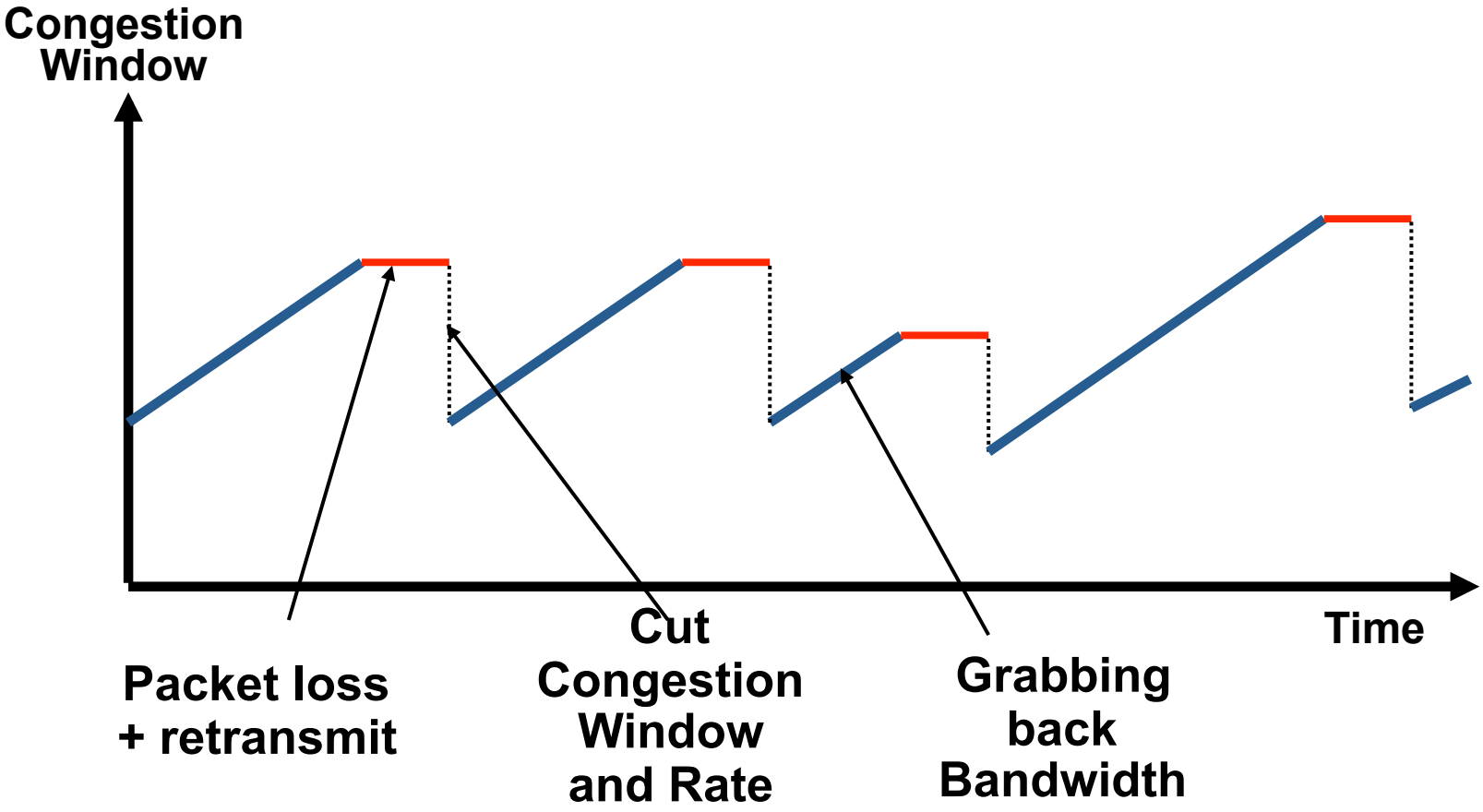
- Upon ACK received:
  - SW+=(1/SW) (increases by ~1 every RTT)
- Upon loss detected:
  - ssthresh = max(SW/2,2), SW=1
  - Revert back to slow start
- Detecting losses
  - Retransmission timeout
  - 3 duplicate ACKs received (fast retransmit)

Estimating RTT – Karn's algorithm

- Estimated RTT (ERRT), based on the observed RTT (ORTT)
  - ERTT (i+1) = α * ERTT(i) + (1-α) * ORTT (i)
  - α typically = 0.875
- Retransmit timer set to 2 * ERTT, increase backoff on retransmission
- Ignore any ORTTs from packets with retransmissions – prevents situations where you are stuck with a erroneously low RTT (http://www.opalsoft.net/qos/TCP-10.htm)

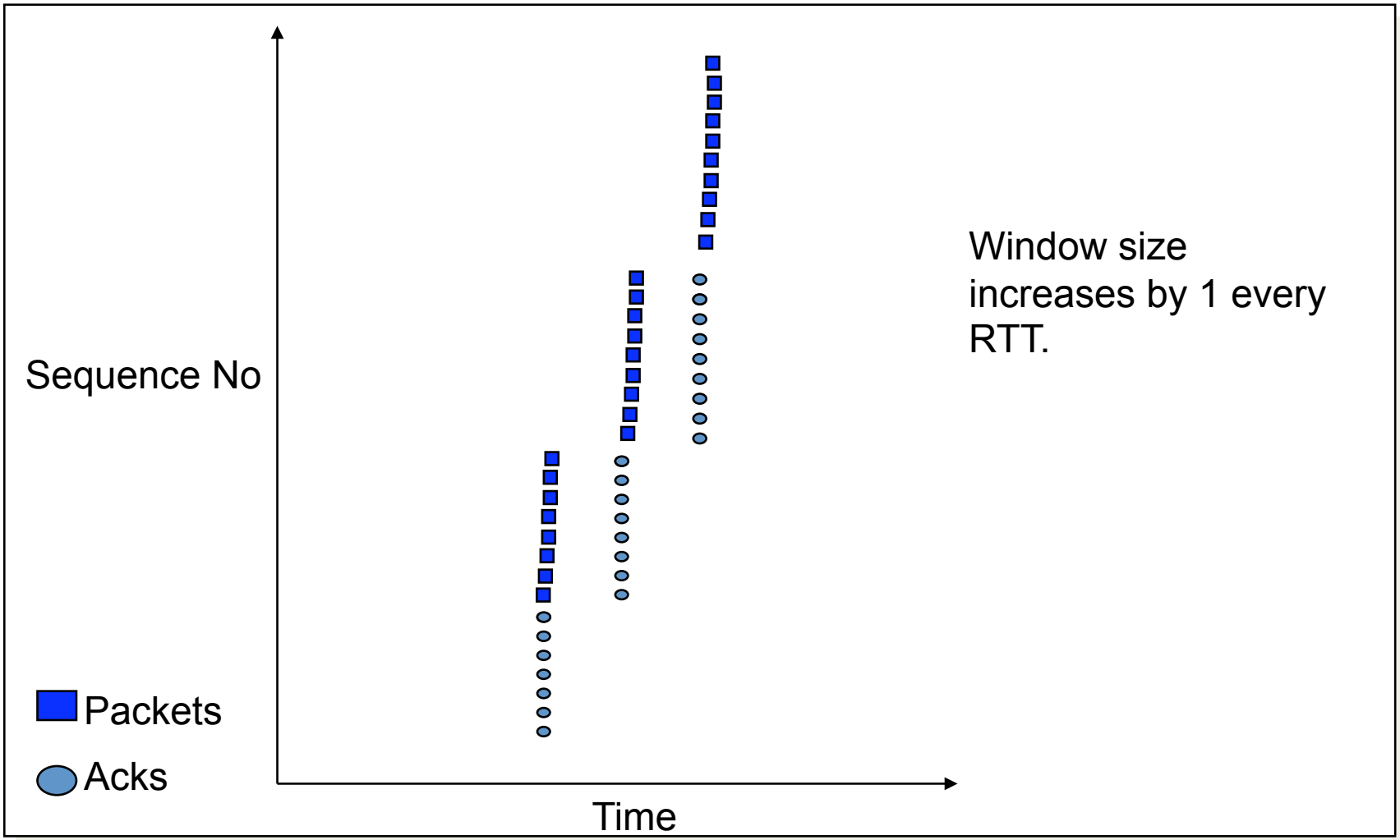
Congestion avoidance - testing

Plot the window size as a function of time for a given flow



Congestion avoidance - testing

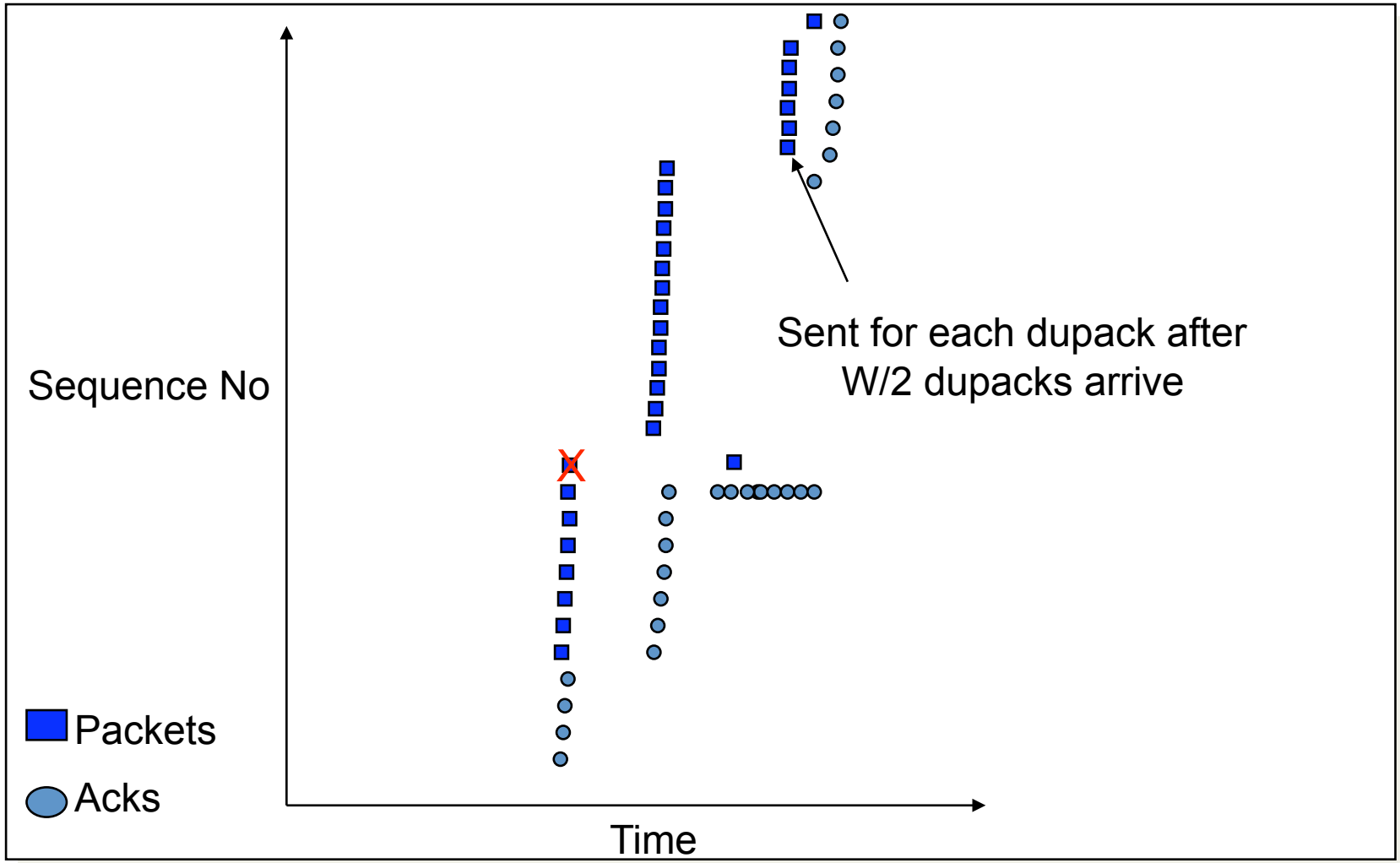
Write time-stamped sequence nos. and ACKS



Fast recovery (optional)

- When a loss is detected, don't go back to slow start, leverage the outstanding packets:
  - Wait to receive SW/2 ACKs
  - Set SW = max(SW/2,2) and resume congestion avoidance

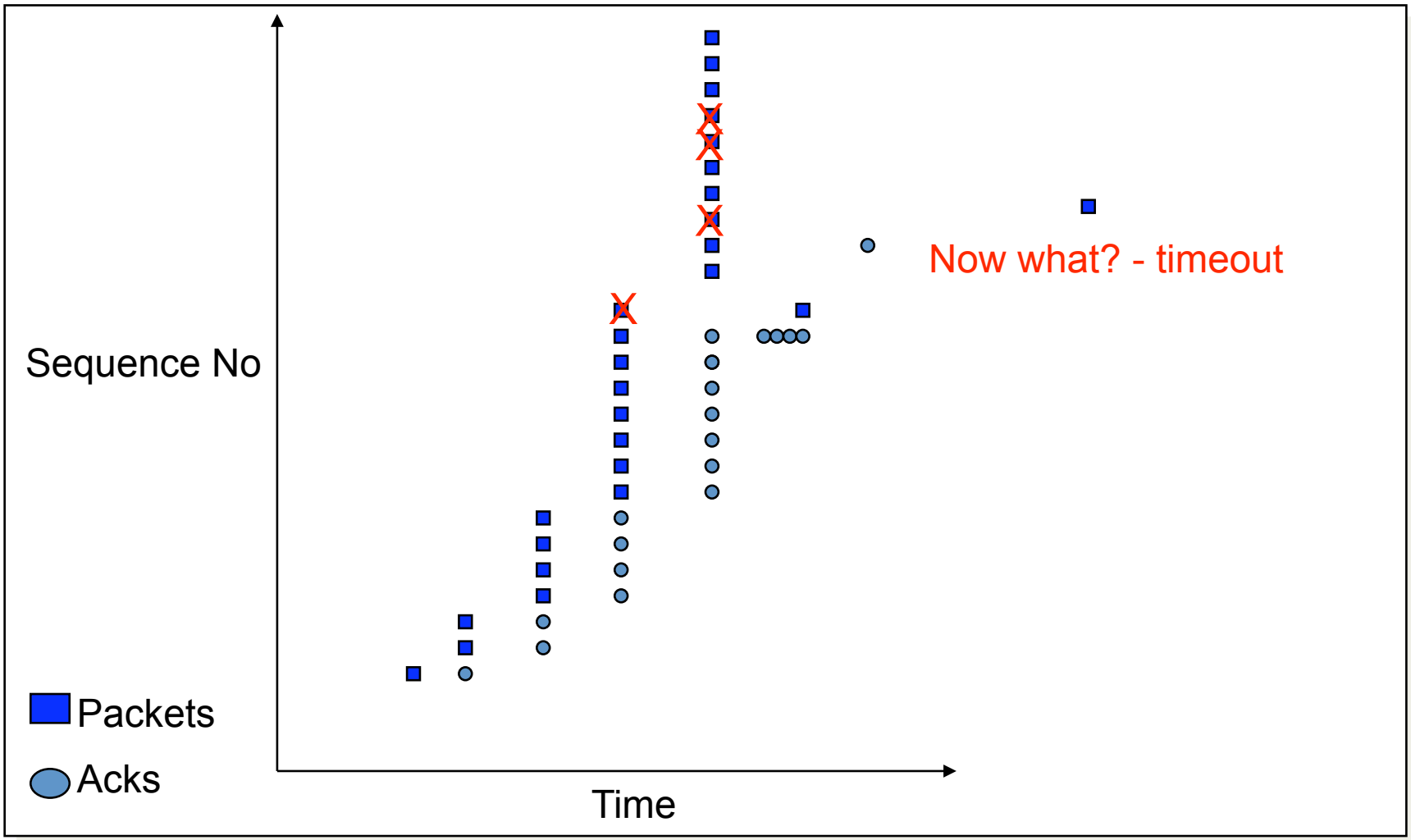
Fast recovery (optional)



Selective ACKs (optional)

- Instead of having the sequence no. of the packet you last received in order
  - send a bitmask of all received packets
- Enables faster retransmissions

Without selective ACKs



With selective ACKs (optional)