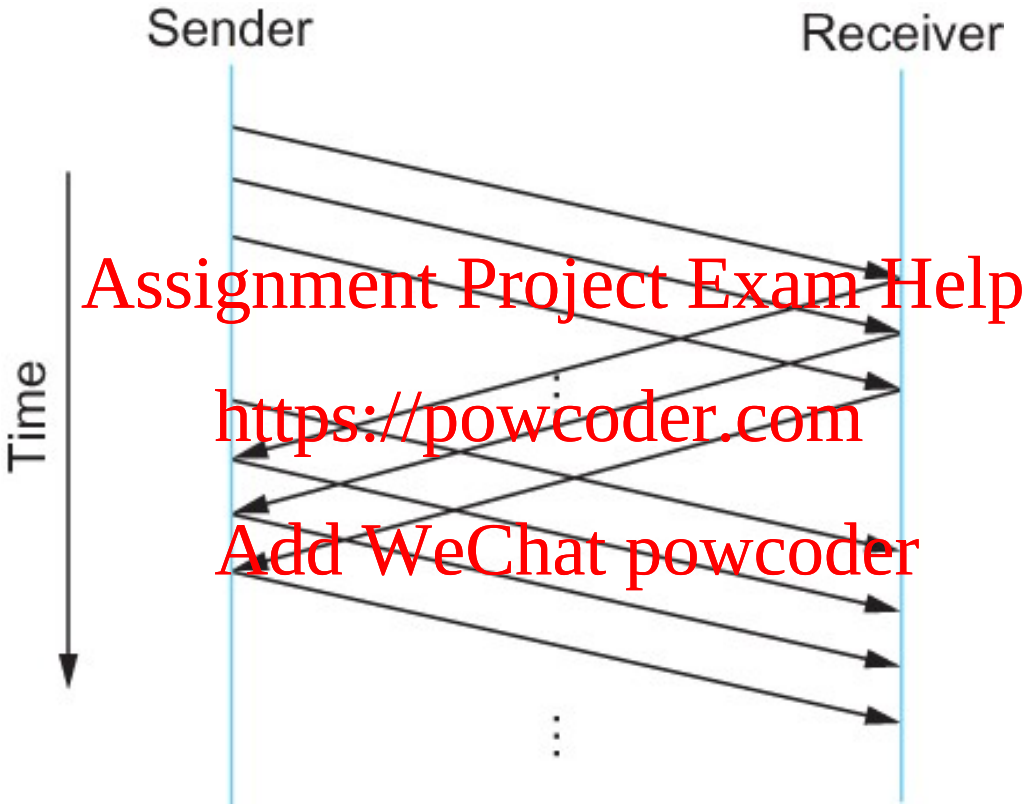
Sliding Window Protocol



Timeline for Sliding Window Protocol

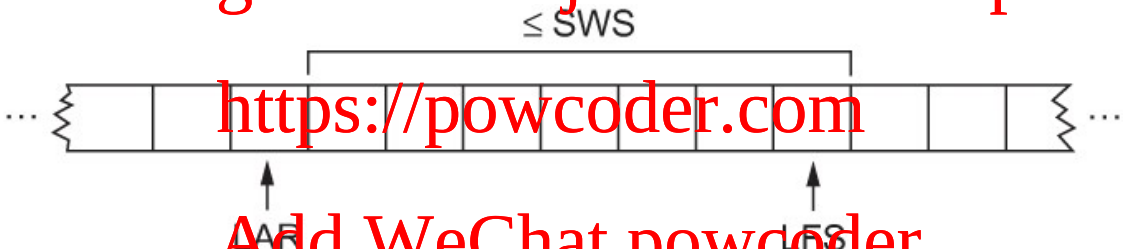
Sliding Window Protocol

- Sender assigns a sequence number denoted as SeqNum to each frame.
  - Assume it can grow infinitely large
- Sender maintains three variables
  - Sending Window Size (SWS)
    - Upper bound on the number of outstanding (unacknowledged) frames that the sender can transmit
  - Last Acknowledgement Received (LAR)
    - Sequence number of the last acknowledgement received
  - Last Frame Sent (LFS)
    - Sequence number of the last frame sent

Sliding Window Protocol

- Sender also maintains the following invariant

$$LFS - LAR \leq SWS$$



Sliding Window on Sender

Sliding Window Protocol

- When an acknowledgement arrives
  - the sender moves LAR to right, thereby allowing the sender to transmit another frame
- Also the sender associates a timer with each frame it transmits
  - It retransmits the frame if the timer expires before the ACK is received
- Note that the sender has to be willing to buffer up to SWS frames
  - WHY?

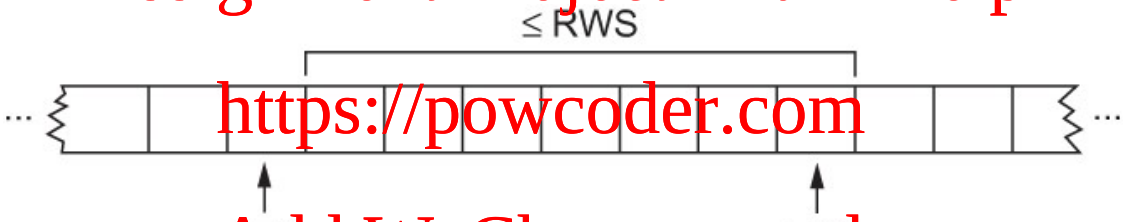
Sliding Window Protocol

- Receiver maintains three variables
  - Receiving Window Size (RWS)
    - Upper bound on the number of out-of-order frames that the receiver is willing to accept
  - Largest Acceptable Frame (LAF)
    - Sequence number of the largest acceptable frame
  - Last Frame Received (LFR)
    - Sequence number of the last frame received

Sliding Window Protocol

- Receiver also maintains the following invariant

$$LAF - LFR \leq RWS$$



Sliding Window on Receiver

Sliding Window Protocol

- When a frame with sequence number SeqNum arrives, what does the receiver do?
  - If SeqNum ≤ LFR or SeqNum > LAF
    - Discard it (the frame is outside the receiver window)
  - If LFR < SeqNum ≤ LAF
    - Accept it
    - Now the receiver needs to decide whether or not to send an ACK

Sliding Window Protocol

- Let SeqNumToAck
  - Denote the largest sequence number not yet acknowledged, such that all frames with sequence number less than or equal to SeqNumToAck have been received

- The receiver acknowledges the receipt of SeqNumToAck even if high-numbered packets have been received

  - This acknowledgement is said to be cumulative.
- The receiver then sets
  - LFR = SeqNumToAck and adjusts
  - LAF = LFR + RWS

Sliding Window Protocol

For example, suppose LFR = 5 and RWS = 4

(i.e. the last ACK that the receiver sent was for seq. no. 5)

⇒ LAF = 9

If frames 7 and 8 arrive, they will be buffered because they are within the receiver window

But no ACK will be sent since frame 6 is yet to arrive

Frames 7 and 8 are out of order

Frame 6 arrives (it is late because it was lost first time and had to be retransmitted)

Now Receiver Acknowledges Frame 8

and bumps LFR to 8

and LAF to 12

Issues with Sliding Window Protocol

- When timeout occurs, the amount of data in transit decreases
  - Since the sender is unable to advance its window
- When the packet loss occurs, this scheme is no longer keeping the pipe full
  - The longer it takes to notice that a packet loss has occurred, the more severe the problem becomes
- How to improve this
  - Negative Acknowledgement (NAK)
  - Additional Acknowledgement
  - Selective Acknowledgement

Issues with Sliding Window Protocol

- Negative Acknowledgement (NAK)
  - Receiver sends NAK for frame 6 when frame 7 arrive (in the previous example)
    - However this is unnecessary since sender's timeout mechanism will be sufficient to catch the situation
- Additional Acknowledgement
  - Receiver sends additional ACK for frame 5 when frame 7 arrives
    - Sender uses duplicate ACK as a clue for frame loss
- Selective Acknowledgement
  - Receiver will acknowledge exactly those frames it has received, rather than the highest number frames
    - Receiver will acknowledge frames 7 and 8
    - Sender knows frame 6 is lost
    - Sender can keep the pipe full (additional complexity)

Issues with Sliding Window Protocol

How to select the window size

- SWS is easy to compute
  - Delay × Bandwidth
- RWS can be anything
  - Two common setting
    - RWS = 1
      No buffer at the receiver for frames that arrive out of order
    - RWS = SWS
      The receiver can buffer frames that the sender transmits

It does not make any sense to keep RWS > SWS

WHY?

Issues with Sliding Window Protocol

- Finite Sequence Number
  - Frame sequence number is specified in the header field
    - Finite size
      - 3 bit: eight possible sequence number: 0, 1, 2, 3, 4, 5, 6, 7
    - It is necessary to wrap around

Issues with Sliding Window Protocol

- How to distinguish between different incarnations of the same sequence number?
  - Number of possible sequence number must be larger than the number of outstanding frames allowed
    - Stop and Wait: One outstanding frame
      - 2 distinct sequence number (0 and 1)
    - Let MaxSeqNum be the number of available sequence numbers
    - SWS + 1 ≤ MaxSeqNum
      - Is this sufficient?

Issues with Sliding Window Protocol

SWS + 1 ≤ MaxSeqNum

- Is this sufficient?

- Depends on RWS
- If RWS = 1, then sufficient
- If RWS = SWS, then not good enough

- For example, we have eight sequence numbers

  0, 1, 2, 3, 4, 5, 6, 7

  RWS = SWS = 7

Sender sends 0, 1, … ,6

Receiver receives 0, 1, … ,6

Receiver acknowledges 0, 1, …, 6

ACK (0, 1, …, 6) are lost

Sender retransmits 0, 1, …, 6

Receiver is expecting 7, 0, …., 5

Issues with Sliding Window Protocol

To avoid this,

If RWS = SWS

SWS < (MaxSeqNum + 1)/2

Issues with Sliding Window Protocol

- Serves three different roles
  - Reliable
  - Preserve the order
    - Each frame has a sequence number
    - The receiver makes sure that it does not pass a frame up to the next higher-level protocol until it has already passed up all frames with a smaller sequence number
  - Frame control
    - Receiver is able to throttle the sender
      - Keeps the sender from overrunning the receiver
        - From transmitting more data than the receiver is able to process