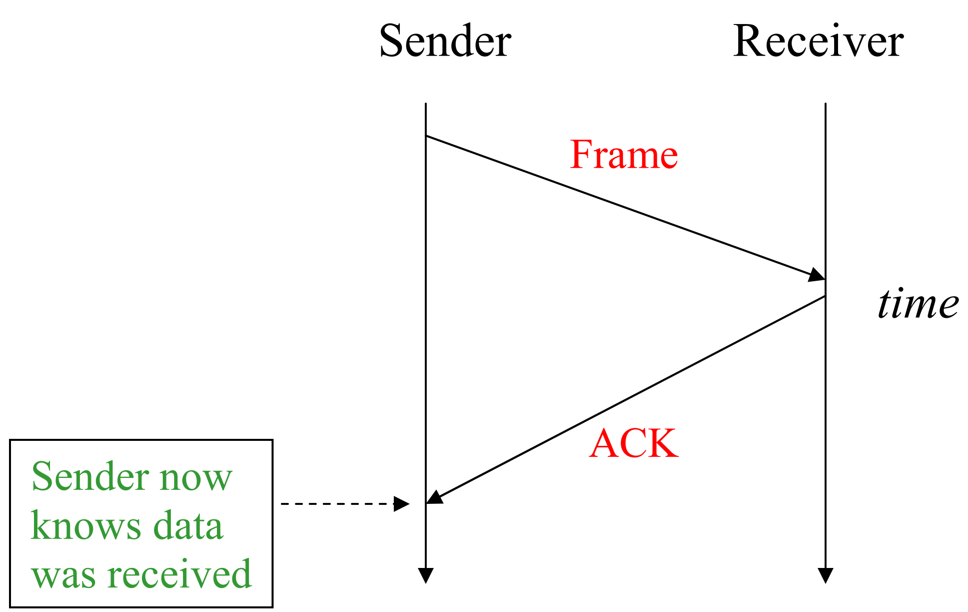
Reliable Transmission

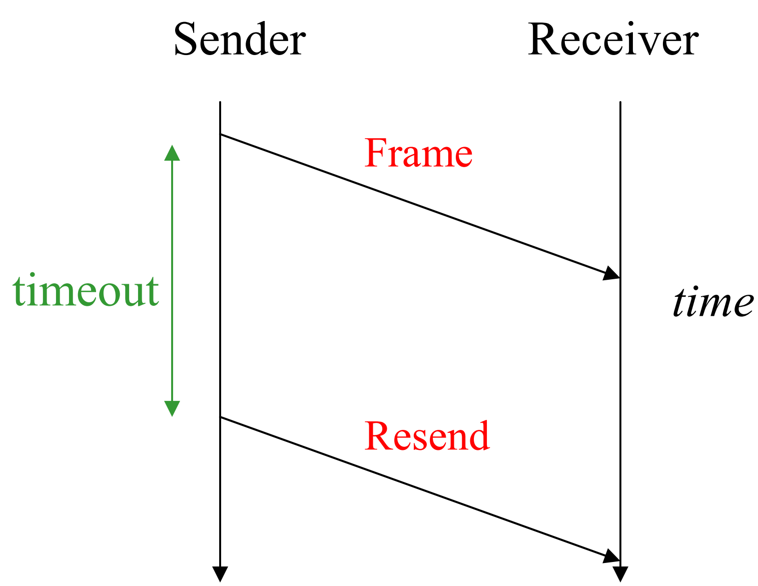
- Because there may be uncorrectable errors (no matter what ECC scheme is used), how can the sender be sure that the receiver got the data?
  - The sender must receive an acknowledgement (ACK) from the sender



What if noACK is received?

Timeouts / Automatic Repeat Request (ARQ)

- If no ACK comes back, the sender must re-send the data (ARQ)
  - When is the sender sure that no ACK is coming back?
    - As a practical matter delays are difficult to bound except for direct links
    - Sender chooses some reasonable <u>timeout</u> – if the ACK isn't back in that much time, it assumes it will never see an ACK, and re-sends



What if original frame arrived, but ACK was lost?

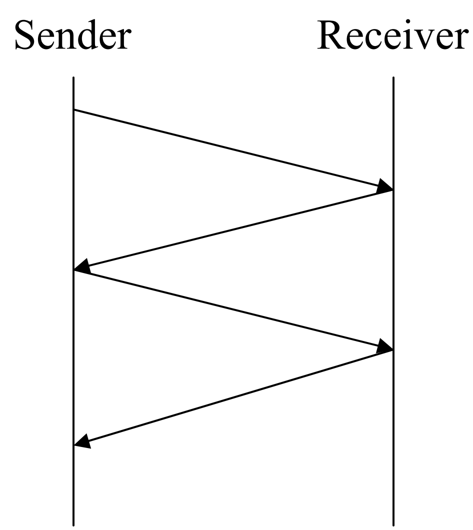
Duplicate Detection: Sequence Numbers

- So that the receiver can detect (and discard) duplicates, distinct frames are given distinct sequence numbers
  - E.g., 0, 1, 2, 3, …

- When a frame is re-sent, it is re-sent with the same sequence number as the original

- The receiver keeps some information about what sequence numbers it has seen, and discards arriving packets that are duplicates

Stop-and-Wait Protocol

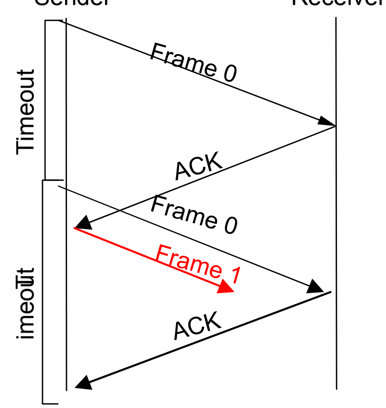


Here's what it looks like when things are going well (no transmission errors).

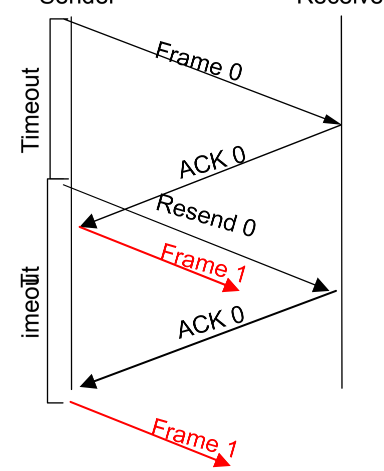
- Sender doesn't send next packet until he's sure receiver has last packet
- The packet/ACK sequence enables reliability
- Sequence numbers help avoid problem of duplicate packets

Stop & wait sequence numbers

- Sequence numbers enable the receiver to discard duplicates
- ACKs must carry sequence number info as well



The Problem Scenario



The Solution

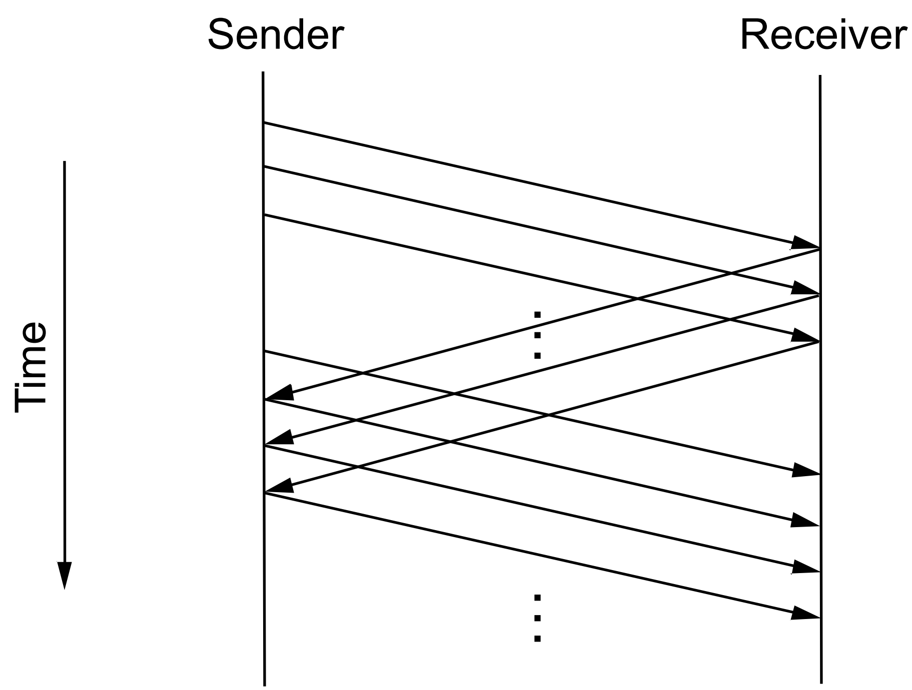
- Stop & wait allows one outstanding frame, requires two distinct sequence numbers

Problem with Stop-And-Wait: Performance

- Problem: “keeping the pipe full”
  - If the bandwidth-delay product is much larger than a packet size, the sender will be unable to keep the link busy
- Example
  - 1.5Mbps link x 45ms RTT = 67.5Kb (8KB)
  - 1KB frames implies 1/8th link utilization
- Solution: allow multiple frames “in flight”

Solution: Allow Multiple Frames in Flight

- This is a form of pipelining



Flow Control

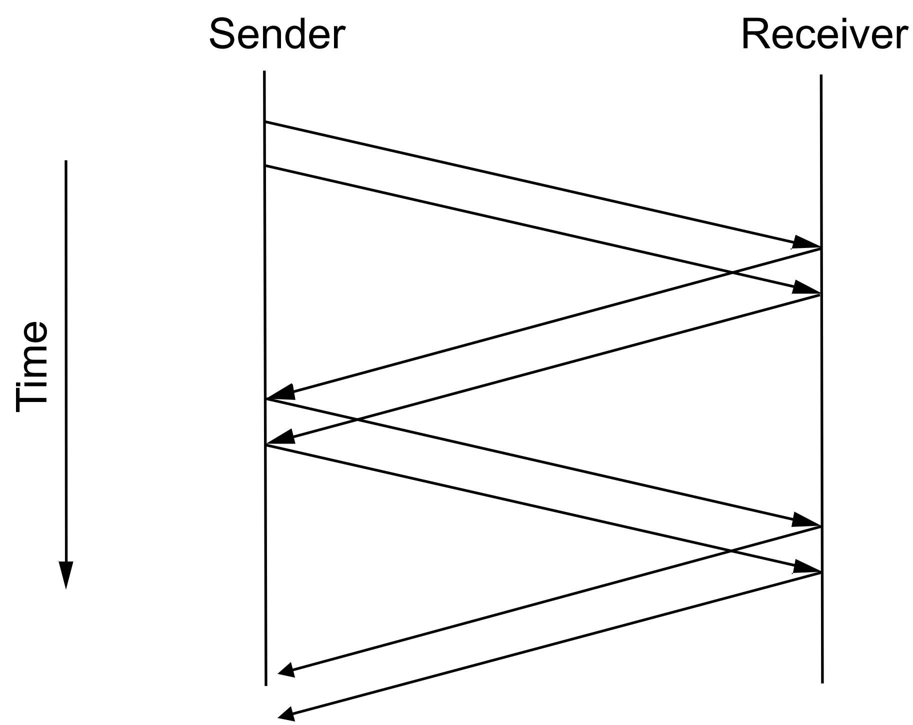
- Why can't we allow the sender to send as fast as it can, timing out and re-sending each frame as necessary?

- Flow control:
  - Receiver needs to buffer data until it can be delivered to higher layers
    - If the sender is much faster than the receiver, it will overwhelm it, causing the receiver to run out of buffer space
  - Additionally, if a frame is lost, the receiver will receive frames "out of order". It wants to buffer those frames to avoid retransmission, but cannot deliver them to the client until the missing frame is re-sent and received
  - Flow control is the notion that the receiver must be able to control the rate at which the sender is thrusting frames at it

- A common approach to flow control is the *sliding window protocol*

Sliding Window Protocol

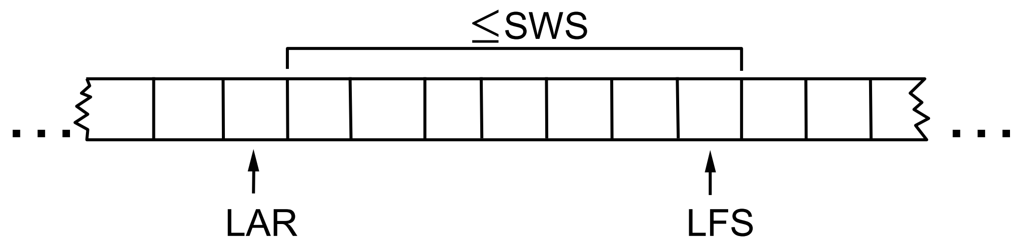
- There is some maximum number of un-ACK'ed frames the sender is allowed to have in flight
  - We call this "the window size"
  - Example: window size = 2



Once the window is full, each ACK'ed frame allows the sender to send one more frame

Sliding Window: Sender

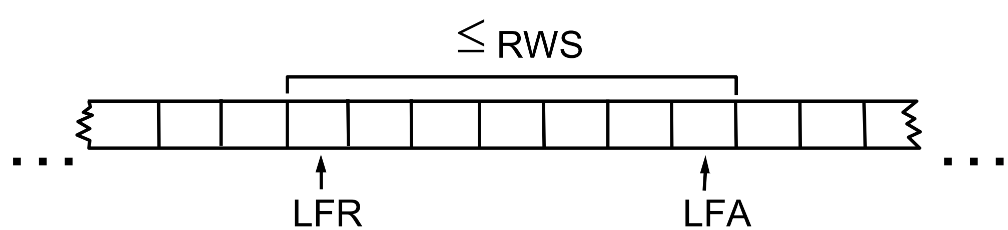
- Assign sequence number to each frame (**`SeqNum`**)
- Maintain three state variables:
  - send window size (**`SWS`**)
  - last acknowledgment received (**`LAR`**)
  - last frame sent (**`LFS`**)
- Maintain invariant: **`LFS`** - **`LAR`** <= **`SWS`**



- Advance **`LAR`** when ACK arrives
- Buffer up to **`SWS`** frames

Sliding Window: Receiver

- Maintain three state variables
  - receive window size (**`RWS`**)
  - largest frame acceptable (**`LFA`**)
  - last frame received (**`LFR`**)
- Maintain invariant: **`LFA - LFR <= RWS`**



- Frame **`SeqNum`** arrives:
  - if **`LFR < SeqNum ≤ LFA`** ⇒ accept + send ACK
  - if **`SeqNum ≤ LFR`** or **`SeqNum > LFA`** ⇒ discard

- Send *cumulative* ACKs – send ACK for largest frame such that all frames less than this have been received

Sequence Number Space

- **`SeqNum`** field is finite; sequence numbers wrap around
- Sequence number space must be larger then number of outstanding frames
- **`SWS <= MaxSeqNum-1`** is not sufficient
  - suppose 3-bit **`SeqNum`** field (0..7)
  - **`SWS=RWS=7`**
  - sender transmit frames 0..6
  - arrive successfully, but ACKs lost
  - sender retransmits 0..6
  - receiver expecting 7, 0..5, but receives the original incarnation of 0..5
- **`SWS < (MaxSeqNum+1)/2`** is correct rule
- Intuitively, **`SeqNum`** "slides" between two halves of sequence number space

Sliding Window Summary

- Sliding window is a well-known algorithm in networking
- First role is to enable reliable delivery of packets
  - Timeouts and acknowledgements
- Second role is to enable in order delivery of packets
  - Receiver doesn’t pass data up to app until it has packets in order
- Third role is to enable flow control
  - Prevents server from overflowing receiver’s buffer