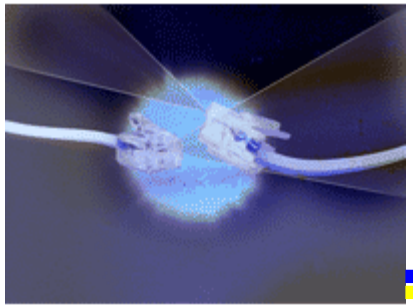
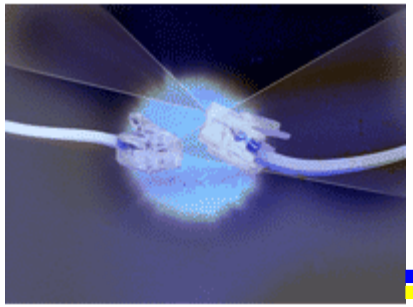


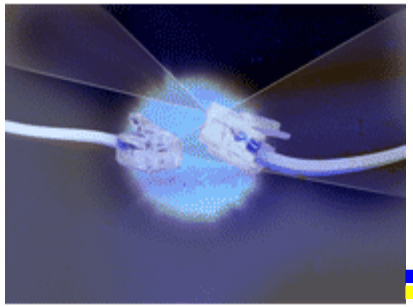
Reliable Transmission

- Desirable for a link-level protocol to deliver frames reliably
  - Error correction is expensive to use, so usually only error detection is used
  - Even if error correction is used, sometimes frames will be unrecoverable
  - Protocol must somehow recover from lost / trashed frames
  - Usually done with a combination of *timeouts* and *acknowledgements*



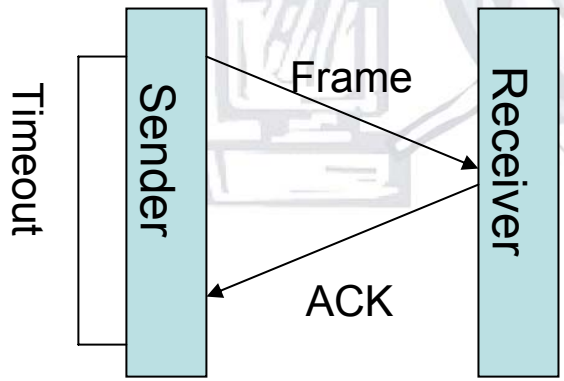
Automatic Repeat Request (ARQ) Mechanism

- Receiver sends short acknowledgement (ACK) to indicate it has received a frame
  - May be able to *piggy-back* the ACK on another frame of data that is being sent back to sender
- If sender doesn't receive ACK within some timeout period, it *retransmits* the frame
- Three different ARQ algorithms
  - Stop-and-Wait
  - Sliding Windows
  - Concurrent Logical Channels

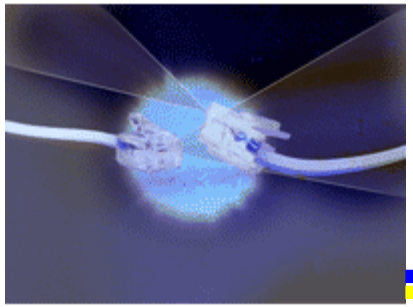


Stop-and-Wait Algorithm

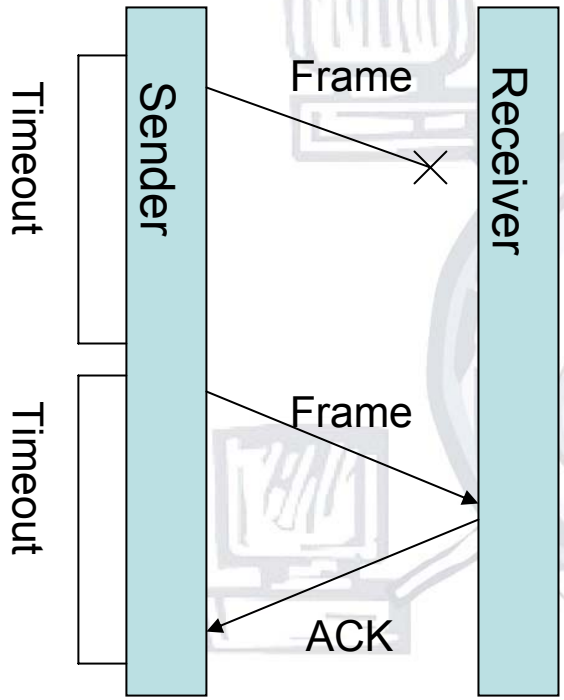
- Sender transmits one frame, starts timer, and waits for acknowledgement
- If timer counts down, sender retransmits
- Four basic scenarios:



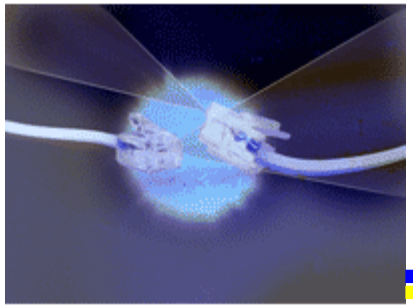
Receiver gets message, sends ACK, and it is received and understood by sender within timeout



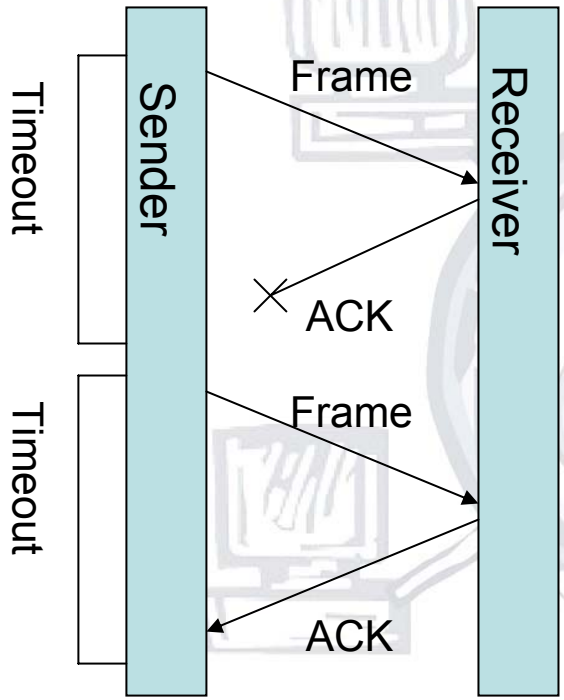
Stop-and-Wait (cont.)

- 

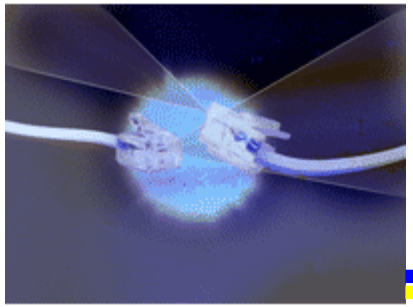
Receiver never gets message. Sender times out, retransmits, and receiver succeeds in returning ACK within timeout



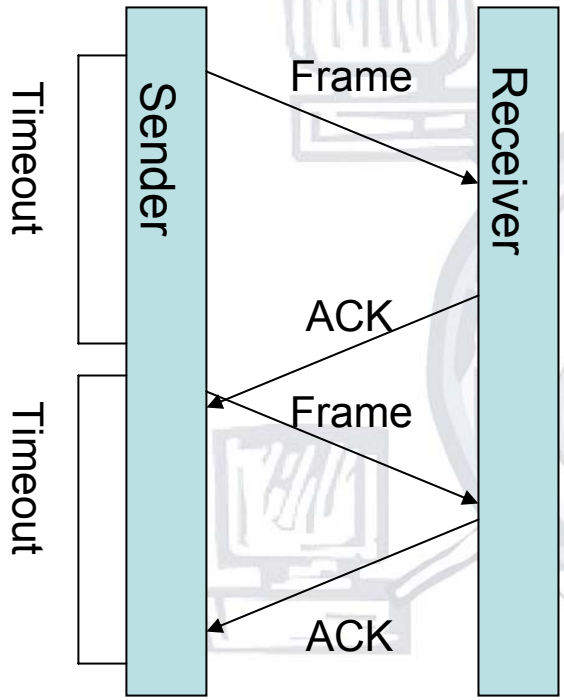
Stop-and-Wait (cont.)

- 

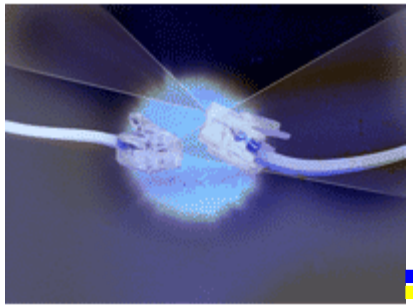
Receiver gets message and ACKs, but ACK is lost. Sender times out, retransmits, and receiver succeeds in returning ACK within timeout.



Stop-and-Wait (cont.)

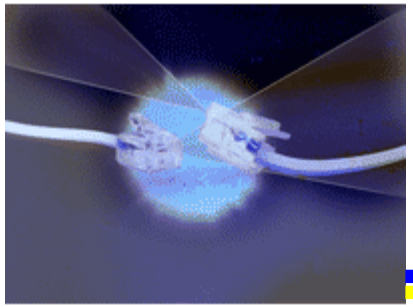
- 

Receiver gets message and ACKs, but ACK does not arrive before timeout. Sender retransmits, and receiver succeeds in returning ACK within timeout.



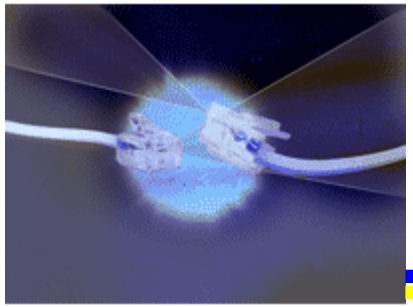
Stop-and-Wait (cont.)

- Receiver might get duplicate copies of message (third and fourth scenarios), so it must be able to identify them
  - Add a one-bit *sequence number* to the frame, so that receiver can tell if a frame is a retransmit or a new one
  - Add the same bit to the acknowledgement - have ACK0 and ACK1, so the sender knows exactly which frame was being acknowledged



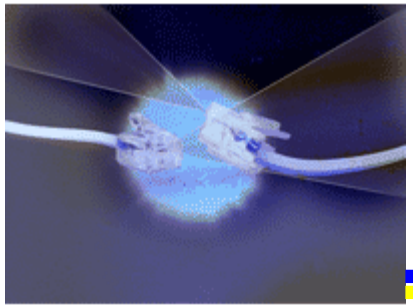
Stop-and-Wait Drawbacks

- Protocol only allows one outstanding frame at a time. If there is high latency on the link, this is extremely inefficient
  - Suppose link has bandwidth of 1 Mbps and RTT of 50 ms.
    Delay x bandwidth = 50 Kb ≈ 6 KB
  - If average frame length is 1 KB, you are only using 1/6 of the link capacity
    You cannot keep the pipe full without having 6 frames outstanding at a time



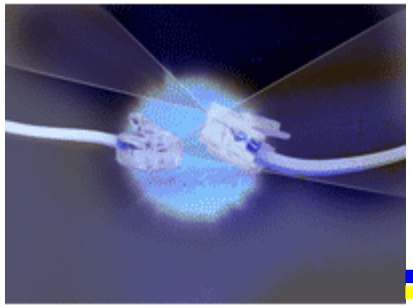
Sliding Window

- Addresses the problem with stop-and-wait by allowing multiple frames to be in transit at the same time
- Each frame is assigned a sequence number (more than the 1-bit number used in stop-and-wait).
- Sender maintains three values:
  - Send Window Size
  - Last Acknowledgement Received
  - Last Frame Sent



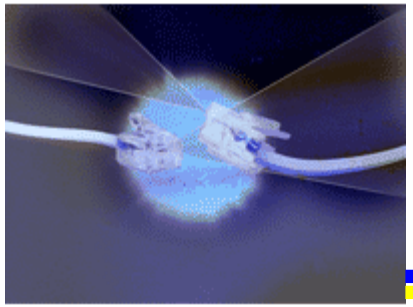
Sliding Window (cont.)

- Require that LFS – LAR ≤ SWS at all times
  - Can't ever have more than SWS unacknowledged frames outstanding
- Once sender has sent SWS frames, cannot send more until one is ACKed and LAR increases



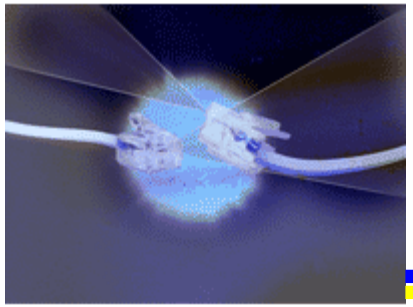
Sliding Window (cont.)

- Receiver also maintains three values:
  - Receive Window Size
  - Largest Acceptable Frame
  - Last Frame Received
- Require that LAF – LFR ≤ RWS always
- Receiver rejects any frame with sequence outside range (LFR, LAF]



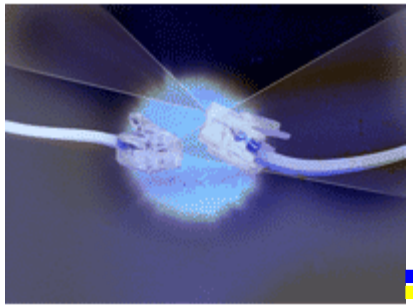
Sliding Window (cont.)

- When receiver gets a good frame, it must decide whether to ACK
  - Should ACK frame with largest seq # such that all frames with smaller seq #s have been received (i.e. the last frame before a hole)
  - Can then bump LAF and move window forward



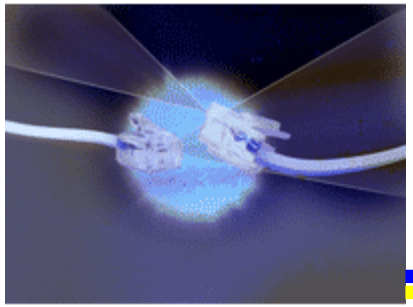
Sliding Window Optimizations

- Desirable to detect packet loss as quickly as possible
  - Send *negative acknowledgement* (NAK) for missing packet when hole in window detected
  - Resend ACK for last frame before hole
  - Use *selective acknowledgements* that can include multiple seq #s
- All of these changes complicate protocol



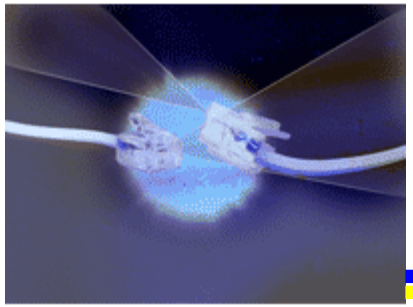
Sliding Window (cont.)

- Window size indicates how many frames sender or receiver must buffer
  - SWS and RWS don't necessarily need to be the same
  - No point making RWS > SWS
- Practical implementation issue
  - Infinitely large seq #s don't fit in a finite number of bits very well



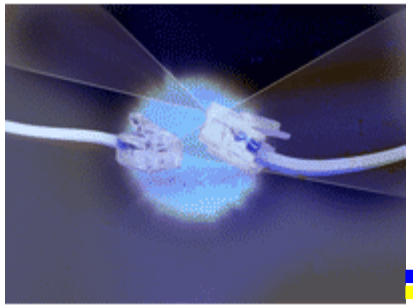
Finite Sequence Numbers

- Limit max seq # to some value (a power of 2 – 1)
- Need SWS < (max seq num + 1) / 2 so that there is no possibility of receiver confusing retransmitted frame with new frame
  - Choose desired SWS based on expected delay x bandwidth and frame size; this will indicate number of bits required for seq #



Frame Order and Flow Control

- Sliding Window Protocol does three things:
  - Reliably deliver frames across link
  - Keep the frames in order
  - Provide *flow control*; allows receiver to control the rate at which sender generates frames
    - Add # frames receiver can take to acknowledged seq # in ACK msg



Concurrent Logical Channels

- ARPANET data link protocol
- Idea is to multiplex several logical channels over single data link, using a stop-and-wait protocol on each logical channel
- Keeps pipe full, but doesn't maintain order, because there is not necessarily any relationship between different channels