CSE 3231
Computer Networks

Chapter 3
The Data Link Layer
part 3 - Flow Control

William Allen, PhD

Spring 2022

Delivering Frames Reliably and Efficiently

- The Data Link layer has to ensure that frames arrive, in order, at the next node
- If a frame has errors, it has to be resent so that the entire message arrives at the destination
- However, it would be inefficient for the Data Link layer to only send one frame and then wait for confirmation before sending another frame
- So, there also has to be a way to *overlap* the transmission and confirmation of frames

Link Reliability

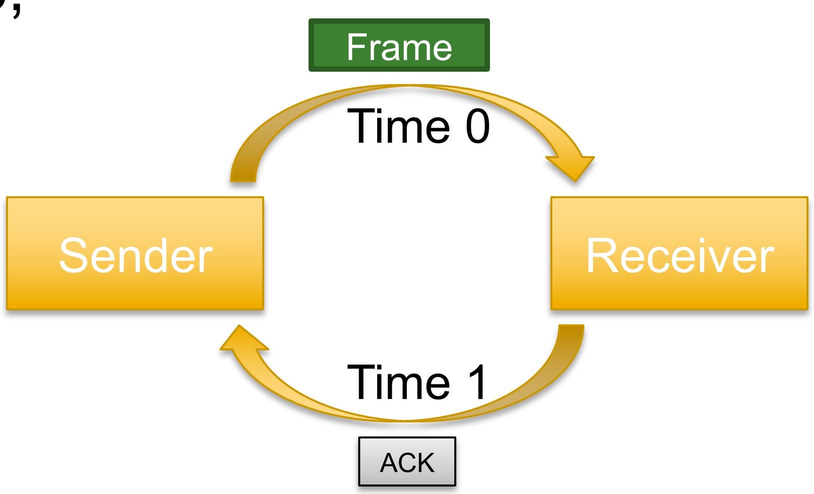
- Normally accomplished through a strategy called Automatic Repeat Request (ARQ), which uses a combination of two fundamental mechanisms
  - Acknowledgement (ACK)
    - an ACK can be transmitted as a *control frame* (e.g. a header without a payload) to reply to a received frame
    - an ACK can also be piggy-backed in data frames
  - Timeout
    - The time period a sender waits for acknowledgement of the reception of an error-free frame -- if no ACK within the timeout period, assume it was lost and resend the frame

ARQ Schemes

- Stop-and-Wait
  - The acknowledgement for each frame must be received *before* the next frame is sent
- Sliding Window
  - Allows for the transmission of *multiple frames* before an acknowledgement is received – to take advantage of the link capacity
    - this is more efficient, but more complicated too

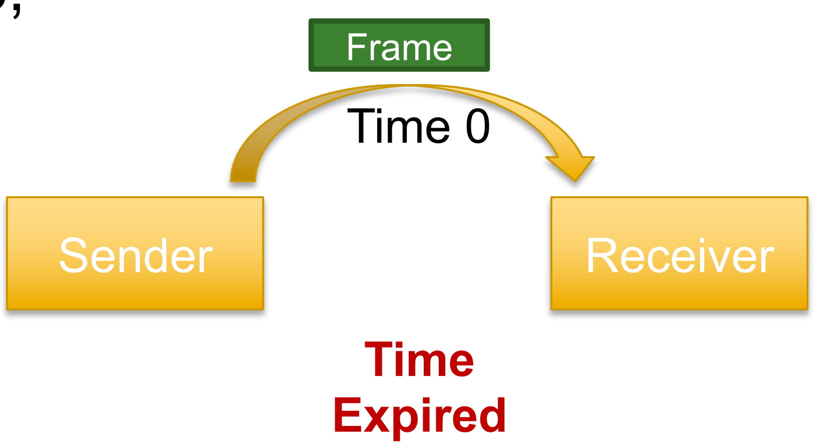
Stop-and-Wait

- After transmitting each frame, the sender **waits** for an acknowledgement from the receiver
- The receiver sends an acknowledgement (ACK) for each frame that is correctly received



Stop-and-Wait

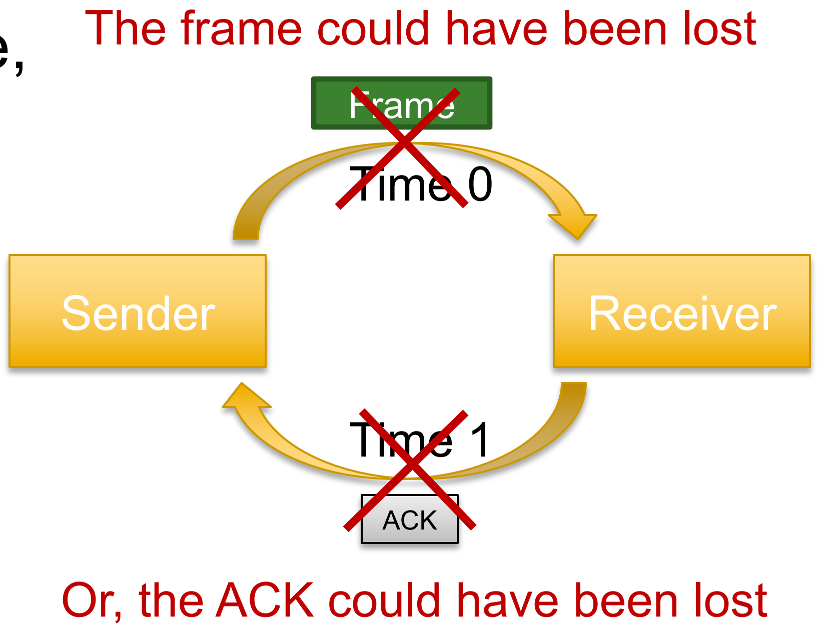
- After transmitting each frame, the sender **waits** for an acknowledgement from the receiver
- The receiver sends an acknowledgement (ACK) for each frame that is correctly received
- If the ACK is not received within the maximum wait interval, the sender **times out** and sends the frame again.



There are two possible causes for this failure.

Stop-and-Wait

- After transmitting each frame, the sender **waits** for an acknowledgement from the receiver
- The receiver sends an acknowledgement (ACK) for each frame that is correctly received
- If the ACK is not received within the maximum wait interval, the sender **times out** and sends the frame again.

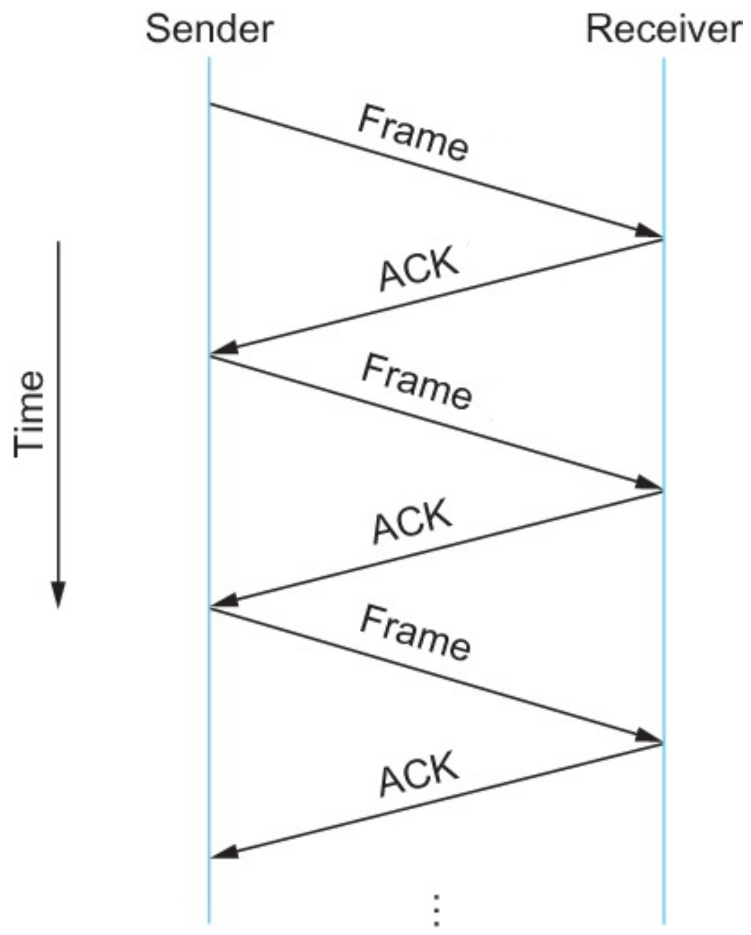

The frame could have been lost
Frame
Time 0
Sender
Receiver
Time 1
ACK
Or, the ACK could have been lost

There are two possible causes for this failure.

Stop and Wait Protocol

Normally, Stop and Wait produces a series of frame deliveries, but only one frame is in transit at a time

- This assumes that a new frame is ready to be sent as soon as the previous frame was acknowledged



Stop and Wait Protocol

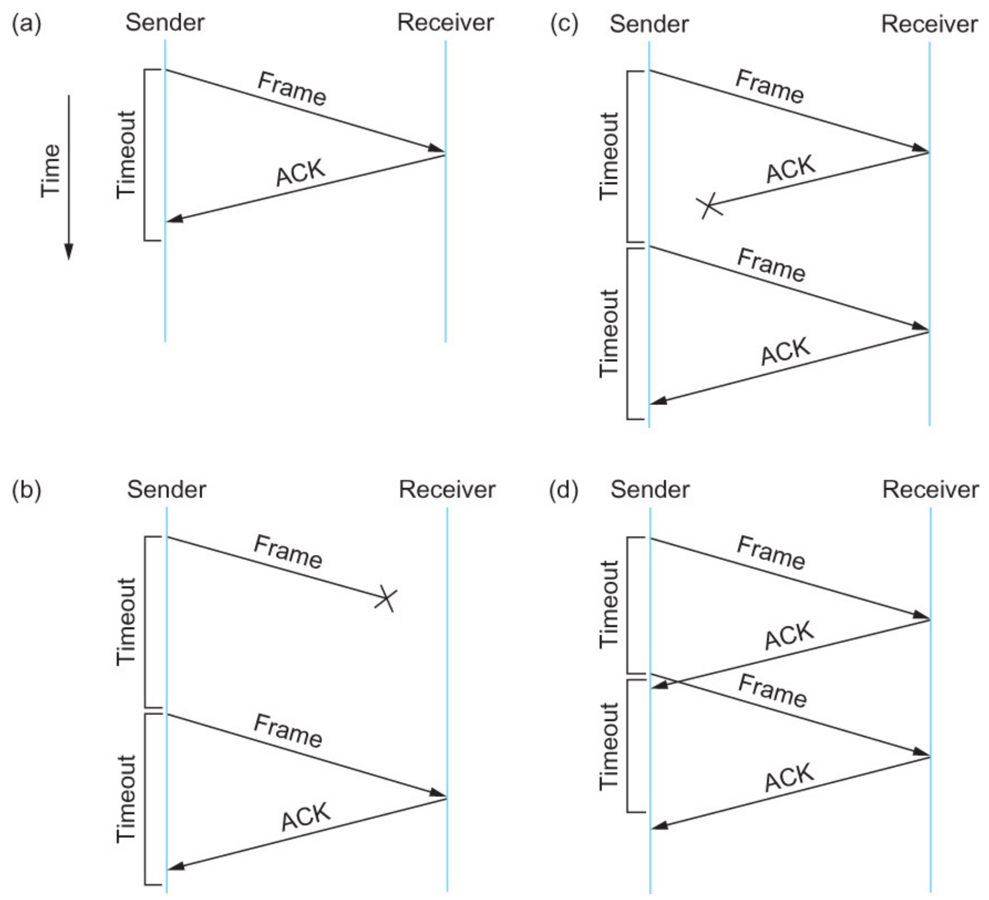
Timeline showing four different scenarios for the stop-and-wait algorithm.

(a) the ACK is received before the timer expires: success

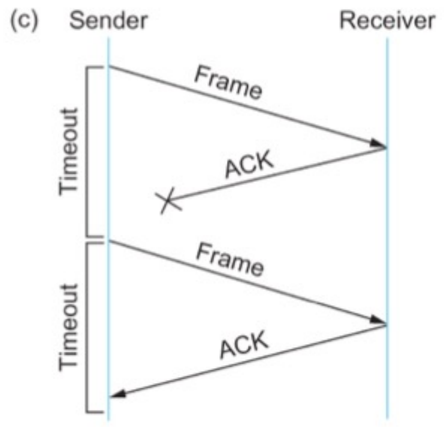
(b) the original frame is lost and must be retransmitted

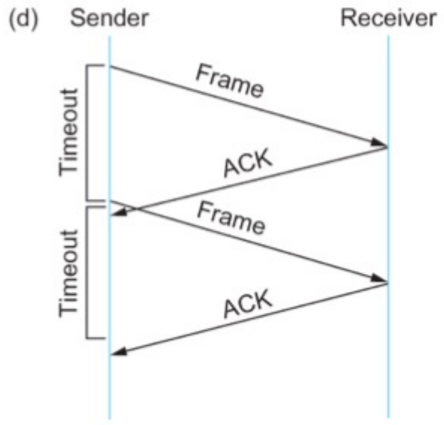
(c) the frame arrives, but the ACK is lost and frame is resent

(d) the timeout occurs too soon and the frame is sent again, even though the frame was received without error



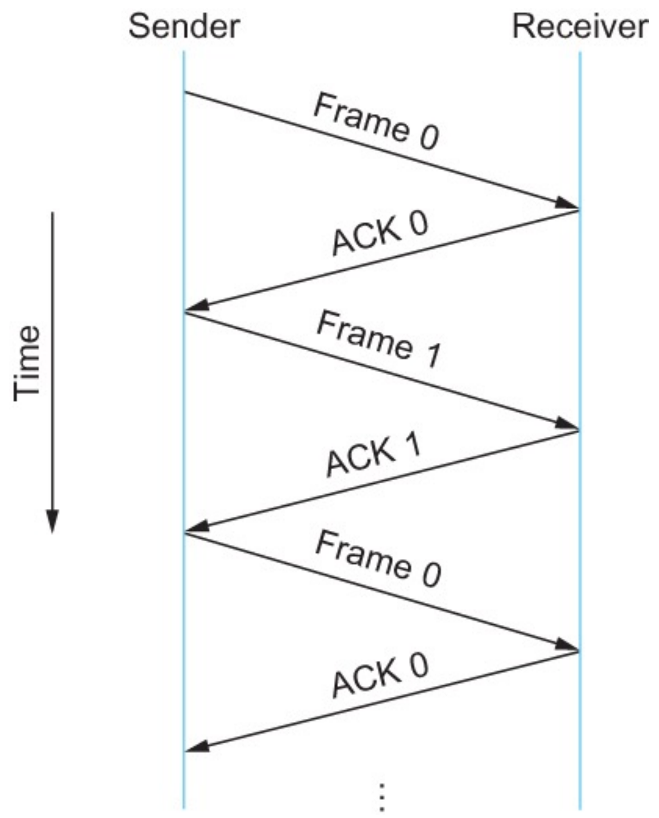
Addressing Duplicates





- Conditions (c) and (d) will result in a packet that had been properly received still being retransmitted
- The receiver cannot differentiate between the two identical frames and will accept the repeated frame as if it were a new frame, causing problems when the higher layers try to process that redundant data

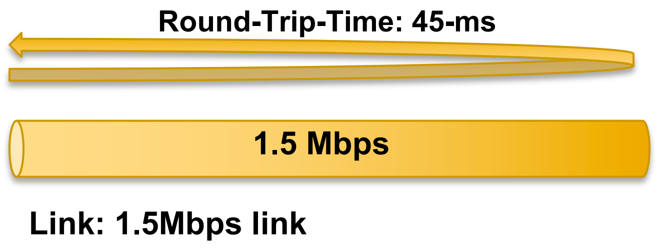
Addressing Duplicates



- To address the problem of duplicate frames, we can add a 1-bit field to the header that will include a sequence number
- Each frame marked with sequence bit = 0 will be followed by a frame with sequence bit = 1 which will be followed by 0, etc.
- Retransmitted frames retain their original sequence number
- This allows the receiver to detect duplicate retransmissions and discard the extra frame

Stop-and-Wait Shortcomings

- If each frame is has to be acknowledged, the sender has to wait for the response before transmitting again
- This delay can lead to under-utilization of the link's capacity



- Consider that 1 KB frames are being sent on the link above
- We can estimate the utilization of the link assuming a stop-and-wait scheme

Stop-and-Wait Shortcomings

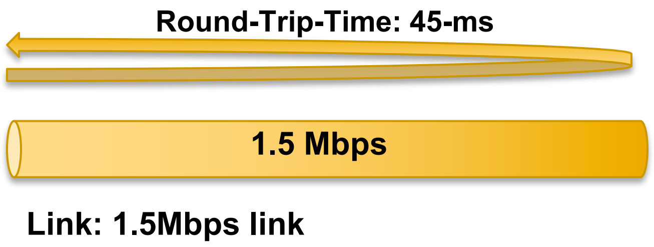
- The *capacity* of this link is:

$$\left(1.5\cdot10^{6}\right)\cdot\left(45\cdot10^{-3}\right)=67.5\cdot10^{3}\ \text{bits}$$

$$67.5\cdot10^{3}\ \text{bits}=67.5\ \text{Kbits}=8.44\ \text{KB}$$

$$\text{Utilization: }1\ \text{K}\ /\ 8.44\ \text{K}\ \approx\ 12\%$$

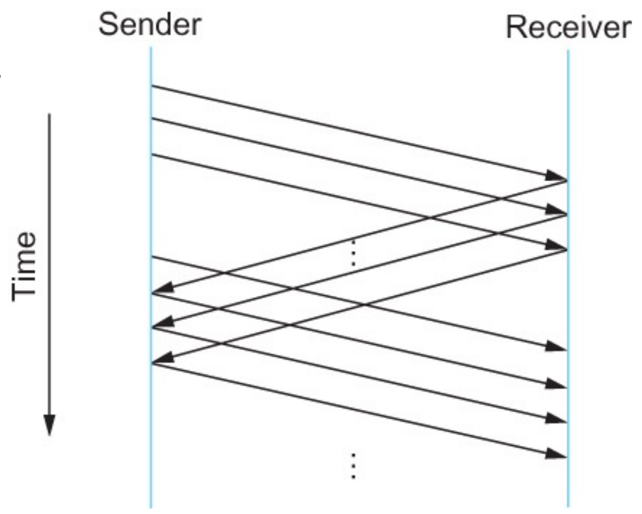
- With Stop-and-Wait, we are only using 12% of the link's capacity
- Since the link's capacity is a little over 8 KB, we could transmit up to eight 1KB frames before the acknowledgment of the first frame arrives from the receiver
- To do that, we will need more than 1 bit to keep track of frames

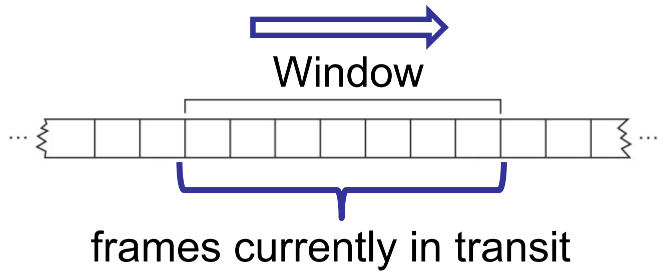


- Consider that 1 KB frames are being sent on the link above
- We can estimate the utilization of the link assuming a stop-and-wait scheme

Sliding Window Protocol

- To increase link utilization, we can send a group of frames and then wait for acknowledgement of those frames before we send additional frames
    - However, we need to keep track of how many frames were sent and how many frames were acknowledged before we can continue transmitting
    - The current group of frames is referred to as a "*window*" and it "*slides*" from one group to another as frames are received and ACK'd





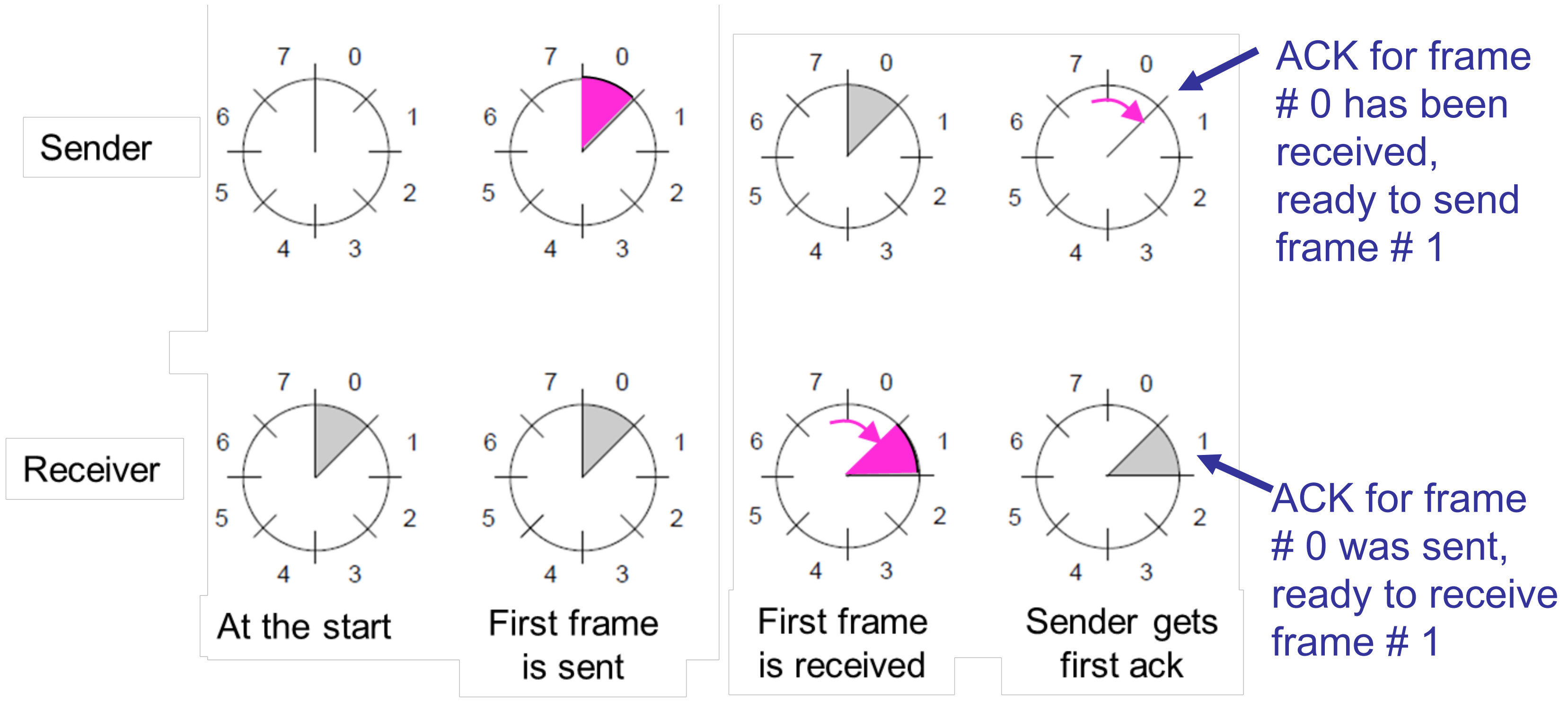
Sliding Window Concepts

- The sending node maintains a window of frames that it *can send*
  - It must *buffer* them for possible retransmission
  - Window advances when acknowledgements are received and new frames can be sent
- The receiving node maintains a window of frames that it *can receive*
  - If a frame is lost, the receiver holds any newer frames in a *buffer* until lost frame is received
  - Checks for errors before sending ACK
  - Window advances with valid in-order arrivals

Sliding Window Concepts

A sliding window advances at the both the sender and the receiver

- Ex: window size: 1, 3-bit sequence number (0-7)

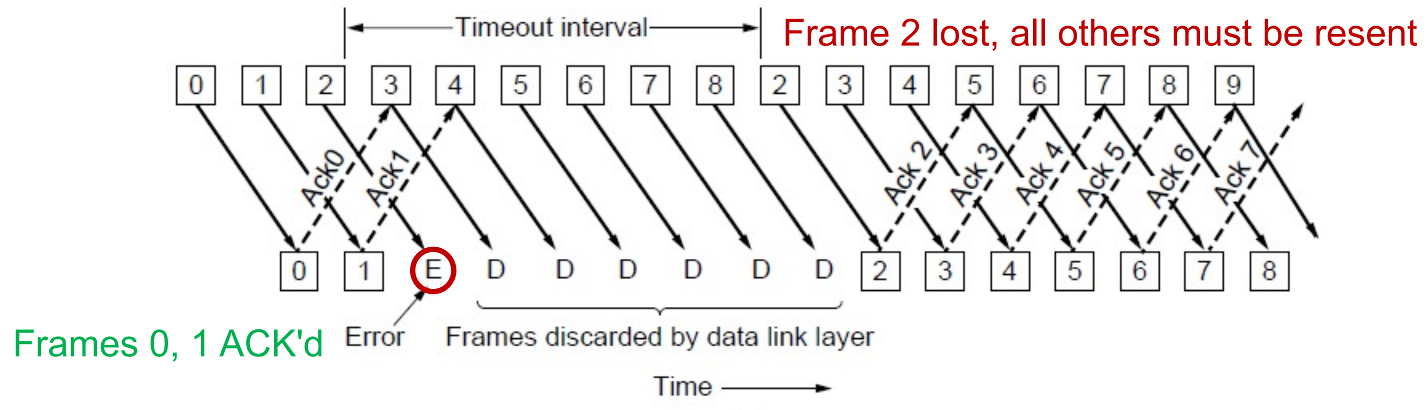


Sender
Receiver
7
0
1
2
3
4
5
6
At the start
First frame is sent
First frame is received
Sender gets first ack
ACK for frame # 0 has been received, ready to send frame # 1
ACK for frame # 0 was sent, ready to receive frame # 1

Go-Back-N

In this version of sliding window, the receiver only ACKs frames that arrive in order:

- It discards frames that follow a missing/error frame
- Sender times out and resends all outstanding frames
- Simple strategy for receiver; buffers only 1 frame
- Wastes bandwidth when using large windows; entire window is retransmitted each time an error occurs



Sliding Window Concepts

Using larger windows enables efficient link use

- Stop-and-wait (window size=1) is clearly inefficient

We need to find the optimum window size

- The best window size (*w*) will depend on the link's capacity, i.e., the bandwidth-delay product (*BD*)
- To take advantage of the full round-trip time we want the window size to be twice that amount: $w \leq 2BD+1$
  - (the +1 is because an ACK isn't sent until the full frame arrives)

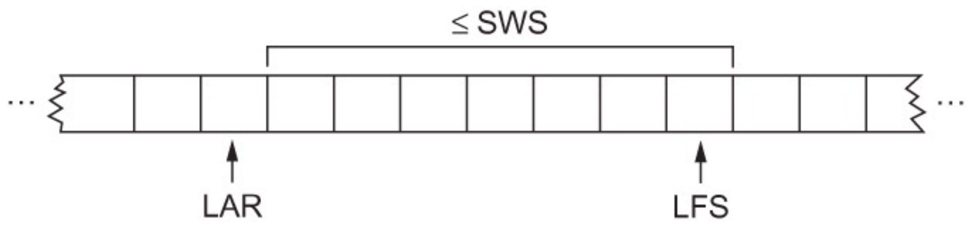
Certain resources are needed to control the flow between nodes and avoid losing track of frames

Sliding Window Protocol

- To keep track of frames, they must each have a unique sequence number that will be carried in the frame's header
    - At some point, the sequence number may reach its maximum value and have to restart at 0
- We need variables at each end of the link to keep track of which frames have been sent and which frames have been acknowledged
    - This is complicated by the possibility that one or more frames in a window may be lost or ACKs for valid frames may not get back to the sender

Sliding Window (Sender Side)

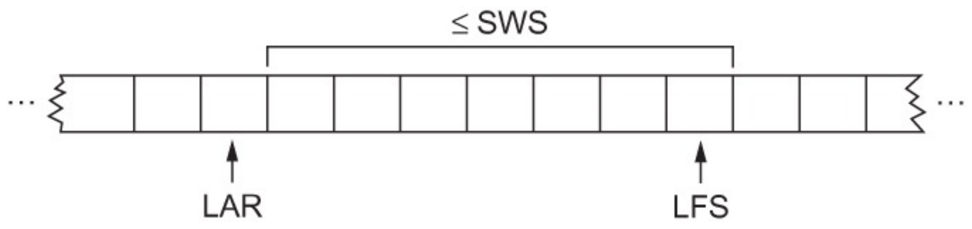
- Each frame is assigned a sequence number
- Three variables are maintained by the **sender**
  - The Send Window Size (SWS)
  - The Last Acknowledgment Received (LAR)
  - The Last Frame Sent (LFS)
- SWS is the maximum number of frames that can be kept waiting for an acknowledgement



LFS – LAR ≤ SWS

Sliding Window (Sender Side)

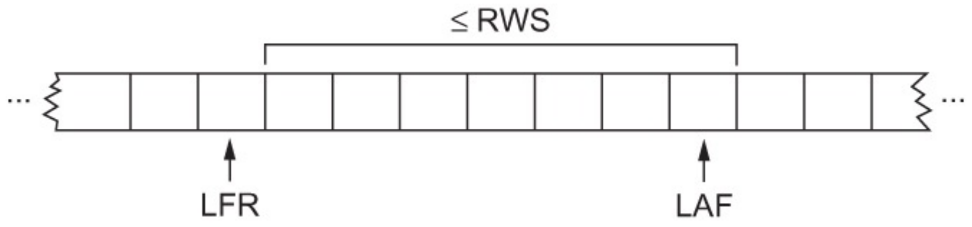
- We start with LAR = LFS = 0 and send frames until LFS = SWS and then pause sending
- As frames are acknowledged, LAR increases and that allows more frames to be sent
  - SWS 'slides' to the right as LAR is incremented, allowing new transmissions which updates LFS
  - If acknowledgements stop, SWS can't advance and no more frames can be sent until an ACK arrives



LFS – LAR ≤ SWS

Sliding Window (Receiver Side)

- Three variables are maintained by the **receiver**
  - The Receive Window Size (RWS)
  - Largest Acceptable Frame (LAF)
    - i.e. highest sequence number the receiver is willing to accept
    - this is to ensure that the input buffer does not overflow
  - The Last Frame Received (LFR)
    - this indicates the highest sequence number that was acknowledged so far, if a frame is discarded due to errors frames that follow it are buffered until that frame is resent



LAF – LFR ≤ RWS

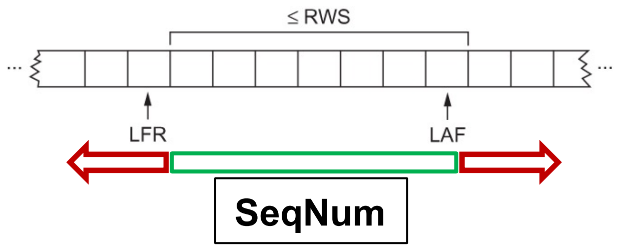
Window Size Selection

- The Sender Window Size (SWS) is normally selected to maximize the utilization of the link (*bandwidth x delay*)
- The Receiver Window Size (RWS) is normally chosen from the following range:

  `1` (i.e., if no buffer) $\leq$ `RWS` $\leq$ `SWS`
- There is no point in selecting RWS greater than SWS, the receiver will never need to maintain more than SWS frames out of order (because the sender can't send more than SWS frames)

Sliding Window

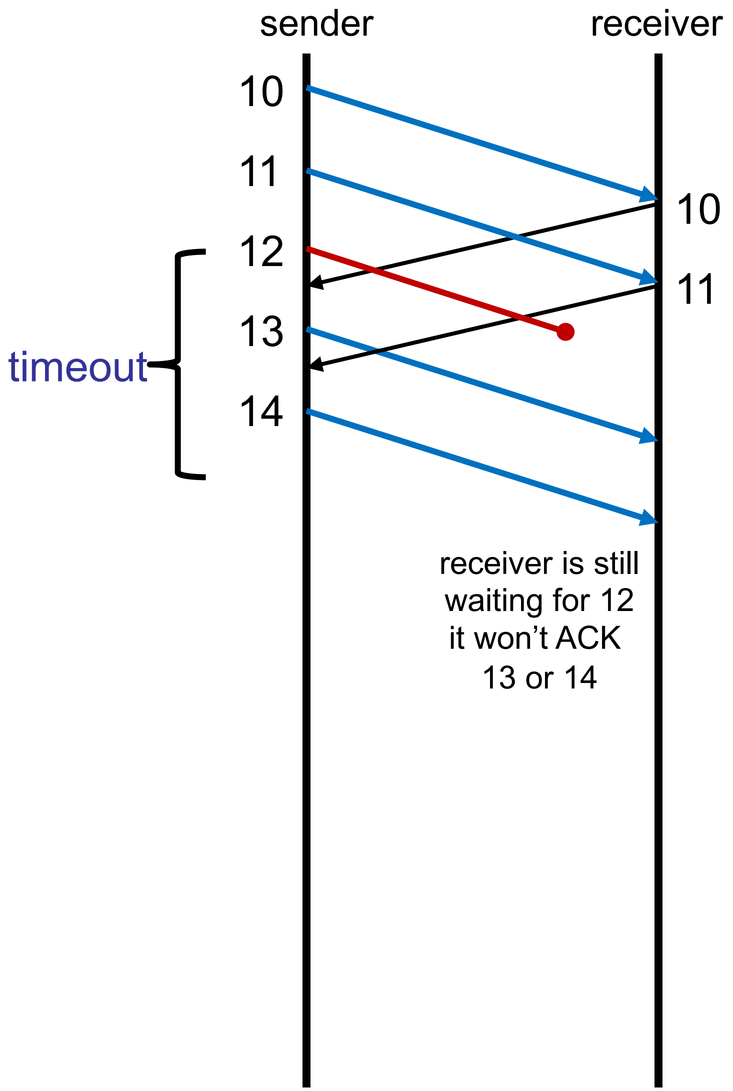
- At the Receiver side, when a Frame is Received with Sequence Number (SeqNum):
  a) SeqNum ≤ LFR → Discard
  b) SeqNum > LAF → Discard
  c) Otherwise Accept



- If a frame arrived with errors, valid frames that follow cannot be ACK'd and are stored in the buffer
- LFR is not advanced past the last frame that was ACK'd
- LAF is advanced to indicate the highest error-free frame
- This allows the receiver to accept frames that arrived after a discarded frame and hold their acknowledgment until the earlier, but still missing frames, arrive

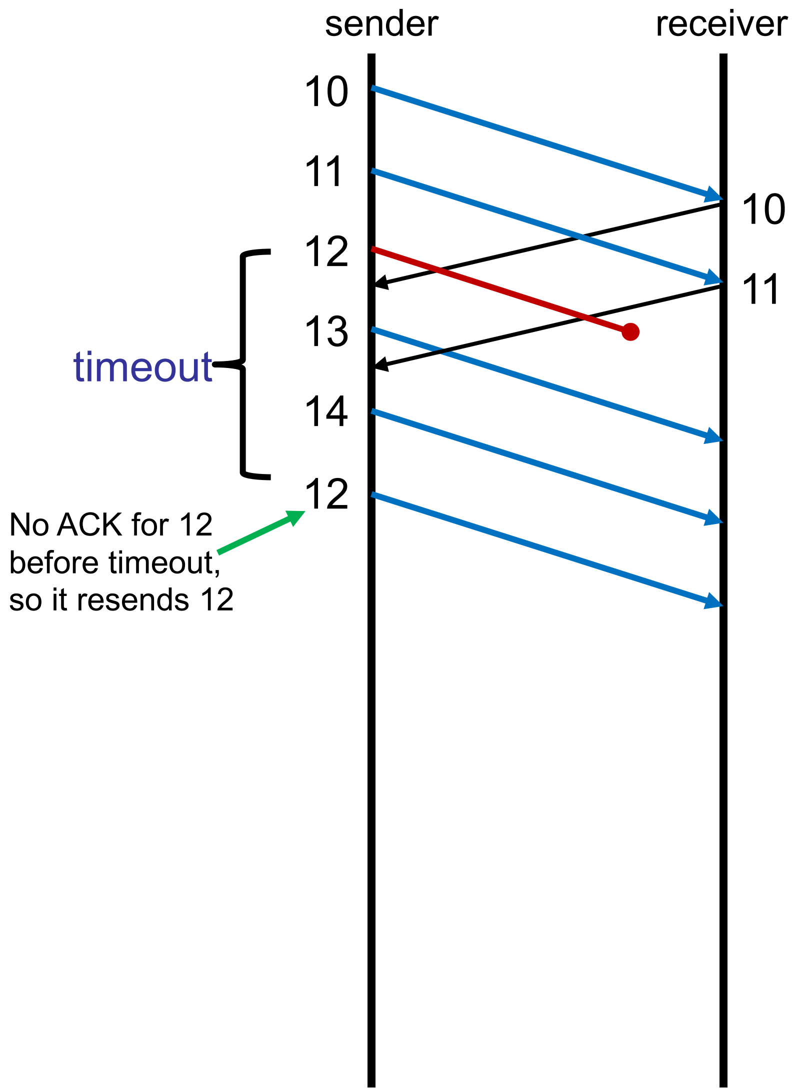
Sliding Window

- Assume that the diagram on the right shows the frames sent and ACK's received
  - Frames are sent every 10ms and the timeout is 29ms
  - There is no delay for processing
  - Incomplete lines are lost and the window is 10 frames
  - Numbers on the left are the frames being sent
  - Numbers on the right are the frames being ACK'd



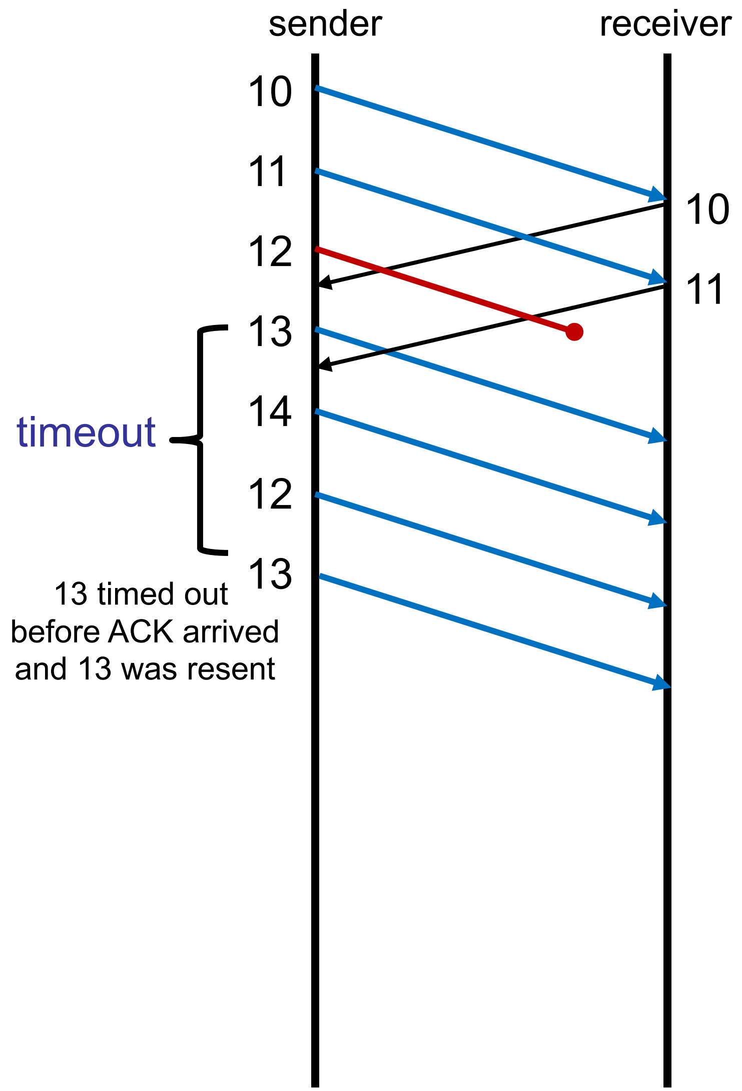
Sliding Window

- Assume that the diagram on the right shows the frames sent and ACK's received
  - Frames are sent every 10ms and the timeout is 29ms
  - There is no delay for processing
  - Incomplete lines are lost and the window is 10 frames
  - Numbers on the left are the frames being sent
  - Numbers on the right are the frames being ACK'd



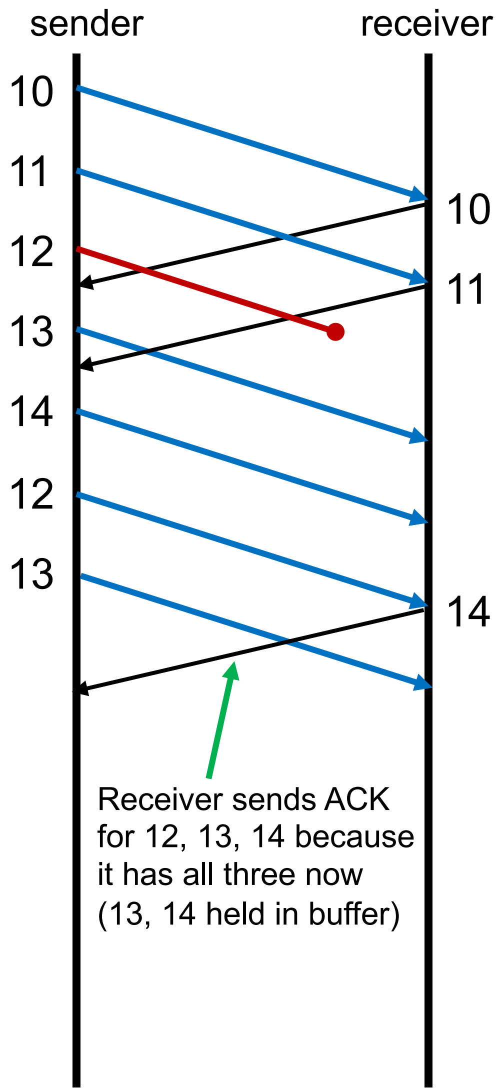
Sliding Window

- Assume that the diagram on the right shows the frames sent and ACK's received
  - Frames are sent every 10ms and the timeout is 29ms
  - There is no delay for processing
  - Incomplete lines are lost and the window is 10 frames
  - Numbers on the left are the frames being sent
  - Numbers on the right are the frames being ACK'd



Sliding Window

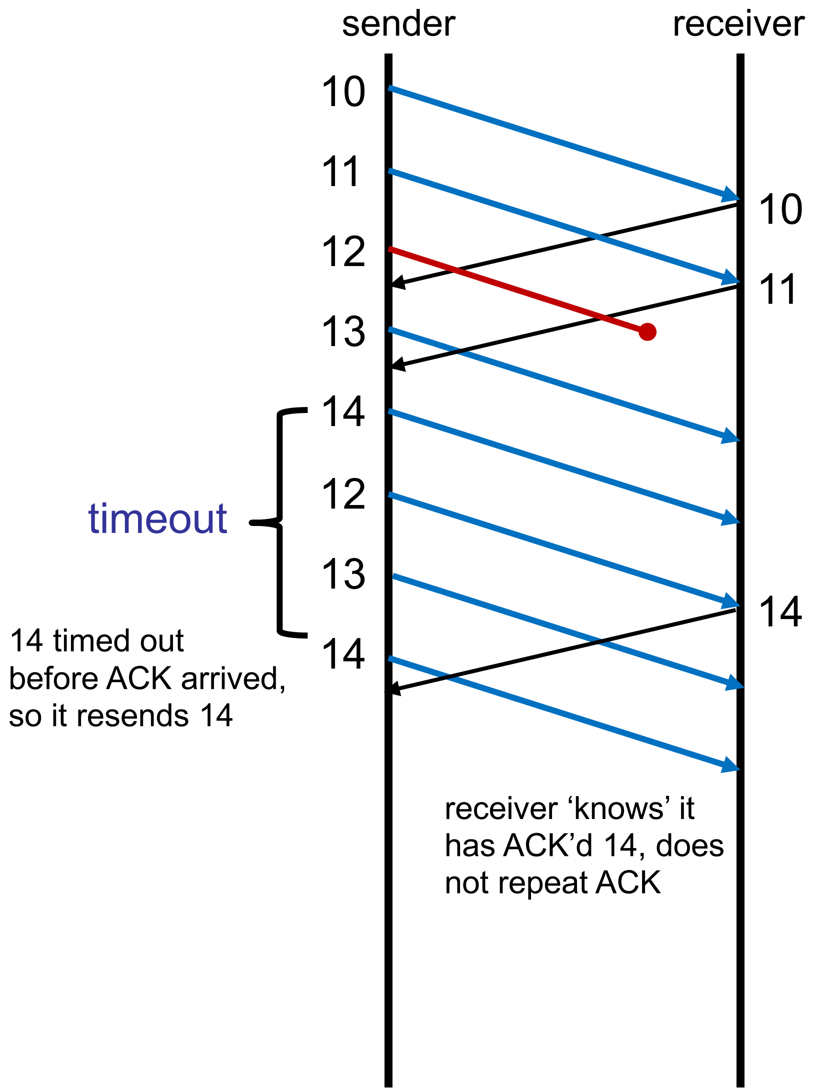
- Assume that the diagram on the right shows the frames sent and ACK's received
  - Frames are sent every 10ms and the timeout is 29ms
  - There is no delay for processing
  - Incomplete lines are lost and the window is 10 frames
  - Numbers on the left are the frames being sent
  - Numbers on the right are the frames being ACK'd



sender
receiver
10
11
12
13
14
12
13
10
11
14
Receiver sends ACK for 12, 13, 14 because it has all three now (13, 14 held in buffer)

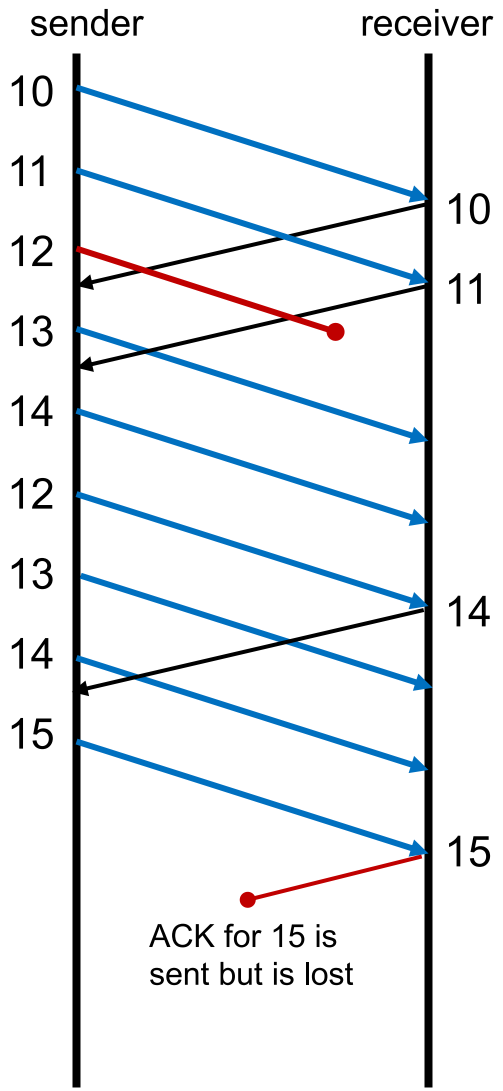
Sliding Window

- Assume that the diagram on the right shows the frames sent and ACK's received
  - Frames are sent every 10ms and the timeout is 29ms
  - There is no delay for processing
  - Incomplete lines are lost and the window is 10 frames
  - Numbers on the left are the frames being sent
  - Numbers on the right are the frames being ACK'd


sender
receiver
10
11
12
13
14
12
13
14
10
11
14
timeout
14 timed out before ACK arrived, so it resends 14
receiver 'knows' it has ACK'd 14, does not repeat ACK

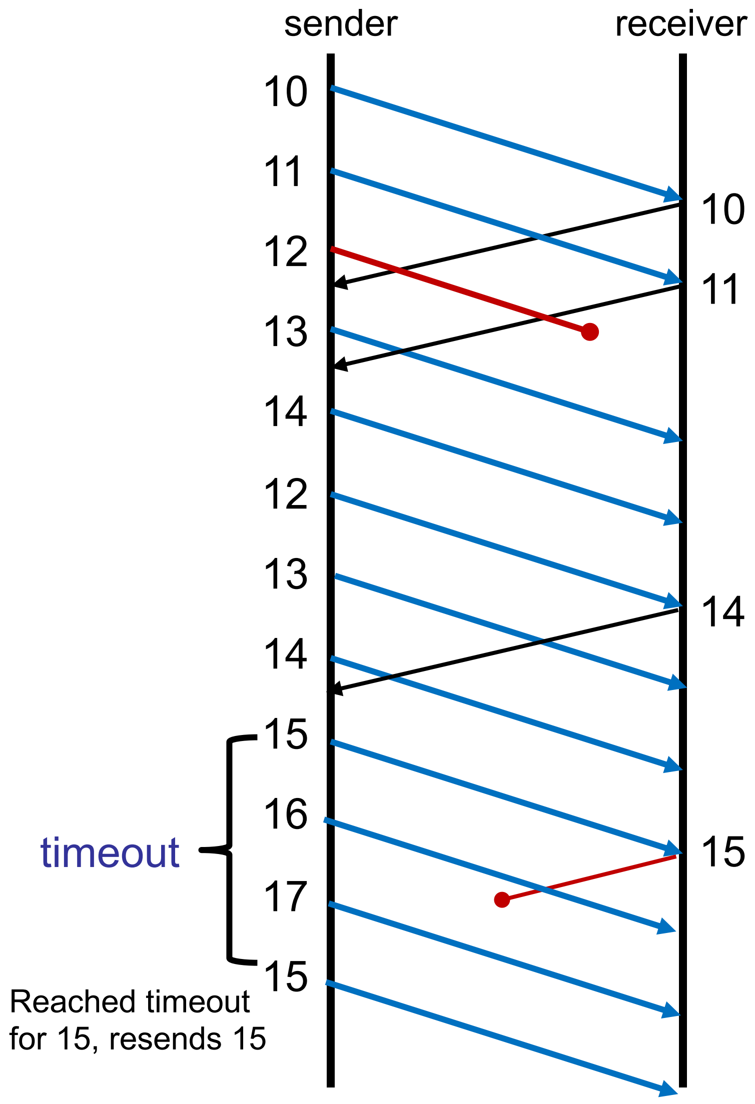
Sliding Window

- Assume that the diagram on the right shows the frames sent and ACK's received
  - Frames are sent every 10ms and the timeout is 29ms
  - There is no delay for processing
  - Incomplete lines are lost and the window is 10 frames
  - Numbers on the left are the frames being sent
  - Numbers on the right are the frames being ACK'd



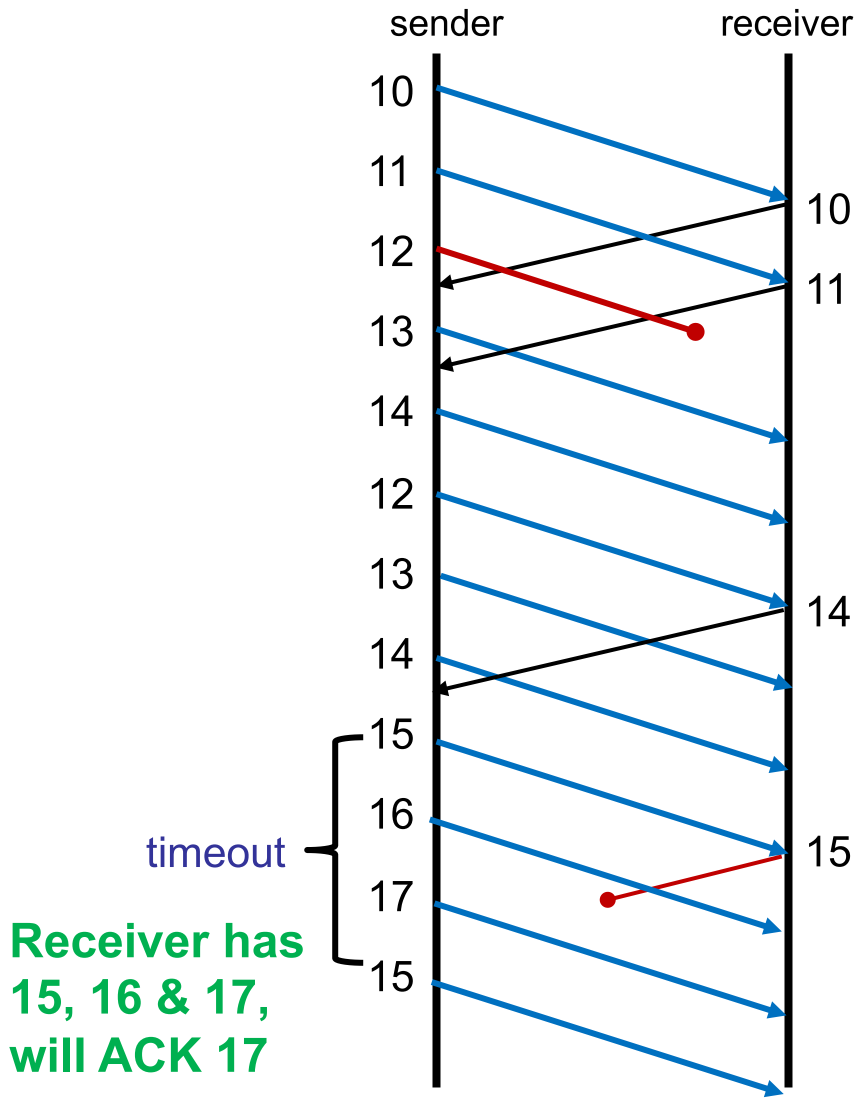
Sliding Window

- Assume that the diagram on the right shows the frames sent and ACK's received
  - Frames are sent every 10ms and the timeout is 29ms
  - There is no delay for processing
  - Incomplete lines are lost and the window is 10 frames
  - Numbers on the left are the frames being sent
  - Numbers on the right are the frames being ACK'd



Sliding Window

- Assume that the diagram on the right shows the frames sent and ACK's received
  - Frames are sent every 10ms and the timeout is 29ms
  - There is no delay for processing
  - Incomplete lines are lost and the window is 10 frames
  - Numbers on the left are the frames being sent
  - Numbers on the right are the frames being ACK'd



Issues with Sliding Window Protocol

- When a timeout occurs, the amount of data in transit decreases
  - Since the sender is unable to advance its window
- When 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
  - Duplicate Acknowledgement
  - Negative Acknowledgement (NAK)
  - Selective Acknowledgement

Variations of the Sliding Window Scheme

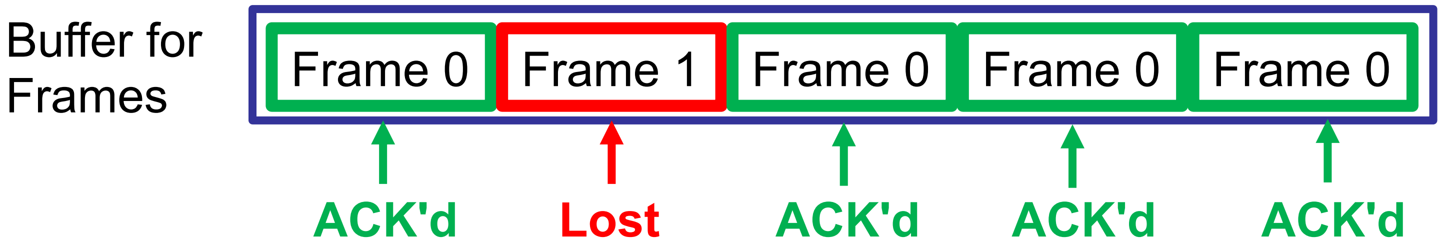
- This assumes frame *k+1* is lost:
  - Duplicate Acknowledgements
    - When the receiver gets frame *k*+2, it could re-send the acknowledgment for frame *k*
    - This duplicate acknowledgement would notify the sender it needs to retransmit the missing frame *k+1*
  - Negative Acknowledgement (NAK)
    - The receiver could send a negative acknowledgement for frame *k+1* as soon as it received frame *k*+2
  - Selective Acknowledgement
    - The receiver could explicitly acknowledge each received frame (*k* and *k*+2), that tells the sender that *k+1* was lost, but it doesn't have to resend *k*+2

Selective Repeat

- Several improvements are combined in a protocol version called Selective Repeat
- The receiver will accept frames that arrive anywhere in the receive window, in any order
  - Cumulative ACK can indicate the frame with the highest in-order sequence number that has been received so far
  - NAK (negative ACK) causes retransmission of a missing frame before a timeout window ends
  - More efficient use of link bandwidth than with Go-Back-N because only lost frames are resent

Selective Repeat

- The receiver sets aside buffer space to hold the frames that were received so that it can assemble the complete message to pass to the Network layer after all frames arrived



- This does increase the resources required to use Selective Repeat, but the reduction in network traffic makes it worthwhile

Dealing with Finite Sequence Numbers

- Sequence Numbers are transmitted as part of each frame, and must be encoded as part of the header
  - A finite number of bits is normally assigned for sequence numbers.
  - There must be an algorithm in place to rotate and reuse sequence numbers
    - clearly, the algorithm will depend on the relative sizes of the sender and receiver windows and is chosen to work best for each specific protocol