Data Link Layer

Flow Control

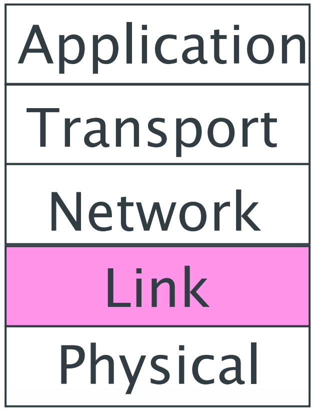
Geoff Merrett

ELEC3227/ELEC6255: Networks

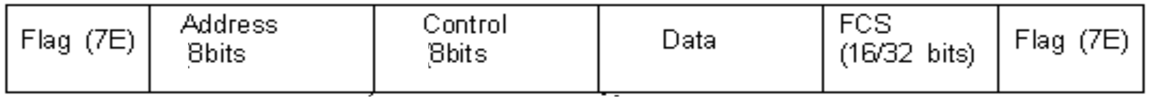
See Tanenbaum Chapter 3 (Data Link Layer)

Outline

- Flow Control
  - Introduction
  - Example Protocols
    - 1: Utopia
    - 2: Stop-and-Wait (Error Free)
    - 3: Stop-and-Wait (Noisy Channel)
    - 4: 1-bit Sliding Window
    - 5: Go-back-N
    - 6: Selective Repeat



Introduction

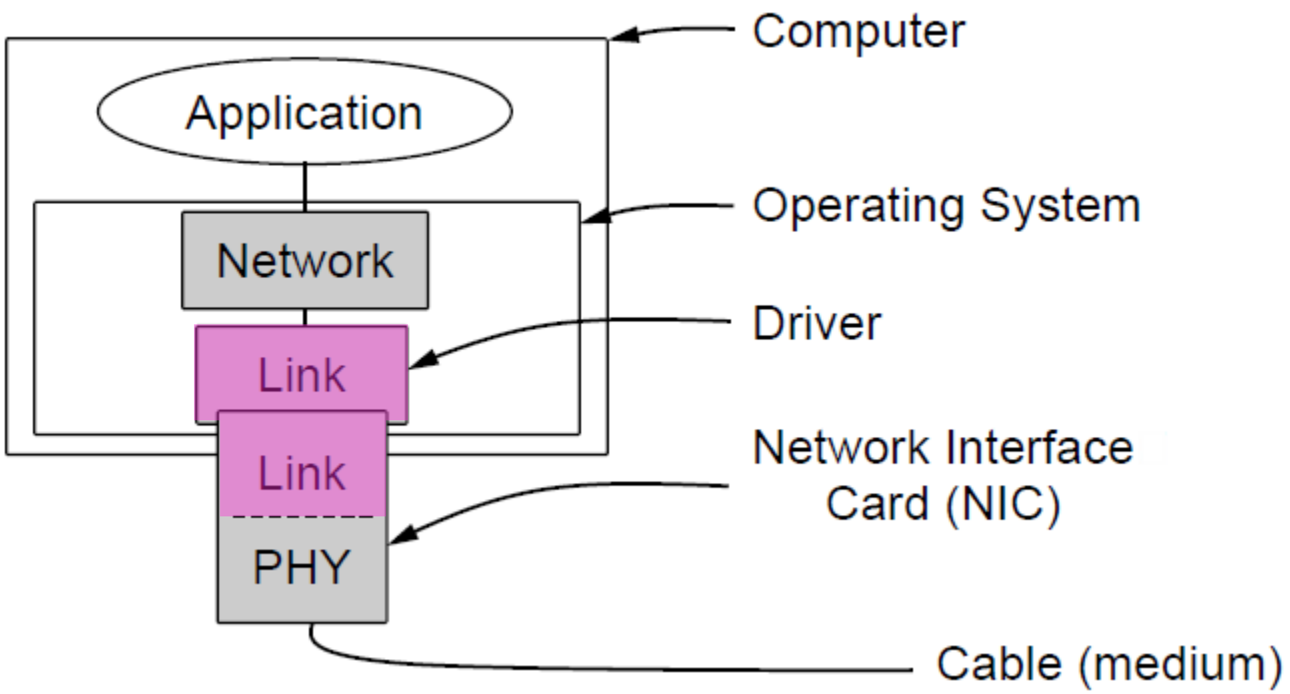


Flag (7E)	Address 8bits	Control 8bits	Data	FCS (16/32 bits)	Flag (7E)

- We know how to:
  - Split a packet up into frames, and delimit their start and end
  - Add error detection into a frame, but…
    - *what do we do when an error is detected?*
  - Add error correction into a frame, but…
    - *what do we do if more errors occur than can be corrected?*
  - *what if the frame is never received (e.g. synchronisation problem)?*
  - *what if the receiver wasn't ready (e.g. transmitting, or processing the last frame)?*
  - we look at the answers to these in this lecture, through a series of examples
- Flow control
  - Prevents a sender out-pacing a slow receiver, by giving feedback on data it can accept
  - Receiver gives feedback on the data it can accept
    - Rare in Link layer as NICs run at "wire speed" (take data as fast as can be sent
  - Flow control is a topic in the Link and Transport layers

DLL in Hardware

- Commonly implemented as Network Interface Card (NIC) and Operating Systems (OS) drivers
- Network layer (IP) is often OS software



Electronics and
Computer Science

UNIVERSITY OF
Southampton

Example Protocols

Example Protocols

- Assumptions
  - The transmitter wants to send a long data stream to the receiver, using a reliable connection-oriented service
  - The transmitter has an infinite supply of data and never has to wait for it to be produced
  - A checksum/CRC is applied in hardware at the PHY/DLL interface (hence we don't consider it in the algorithms)
  - The machines don't crash!
- Process
  - The DLL receives packets from the NET; encapsulated into a frame adding header and footer, and passed to the PHY.
  - The DLL receives packets from the PHY; checked, 'de'-encapsulated and passed to the NET.
- Notes
  - Video examples are on Blackboard (contact me if you can't find them)
  - Look at the textbook for more info (and pseudocode examples)

P1: Utopia

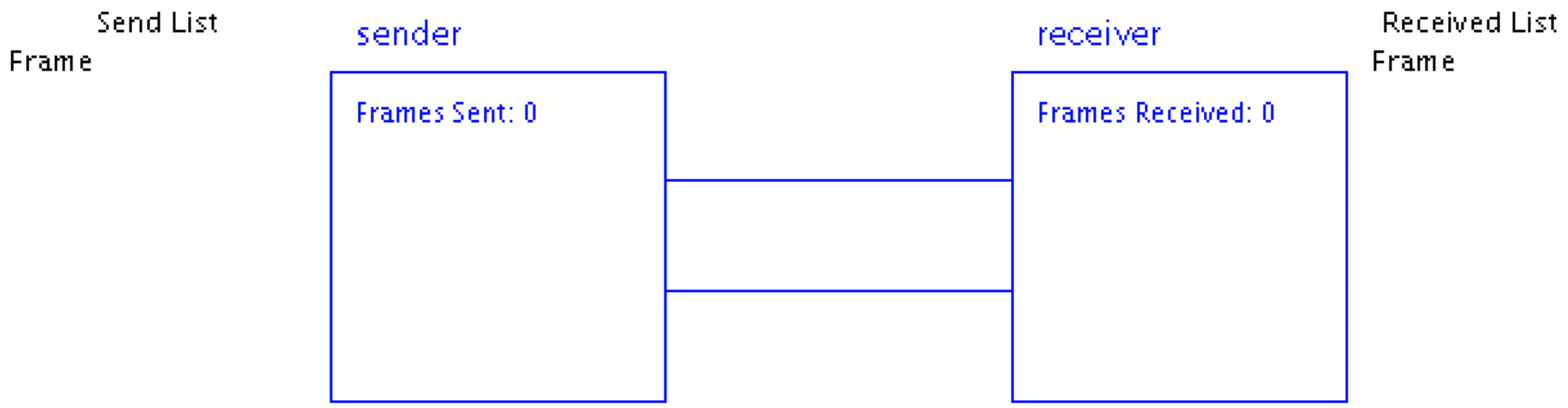
- An optimistic protocol as a starting point
  - Assumes no errors, data always available, receiver as fast as sender etc
  - Considers one-way data transfer
  - Like an unacknowledged connectionless service; assumes higher layers handle all!
  - That's it, no error or flow control …

DEMO!

UNIVERSITY OF Southampton

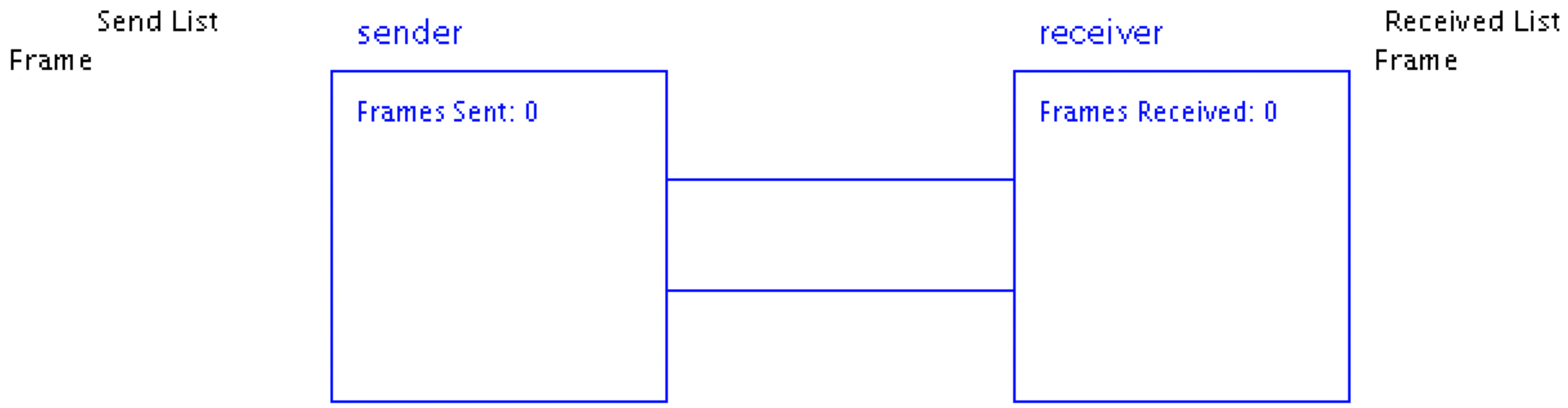
P1: Utopia (*illustration*)

- **Protocol 1 assumes that no errors occur and that the receiver can handle frames as fast as they are produced.**
- *Scenario 2: In this scenario, 3 frames are sent with a second between them.*



P1: Utopia (*illustration*)

- **Protocol 1 assumes that no errors occur and that the receiver can handle frames as fast as they are produced.**
- *Scenario 3: In this scenario, 3 frames are sent with a second between them. The second frame sent is lost and the protocol fails. The receiver thinks that it has received frames 0 and 1 but in reality it received frames 0 and 2. The second frame has been lost and cannot be recovered.*



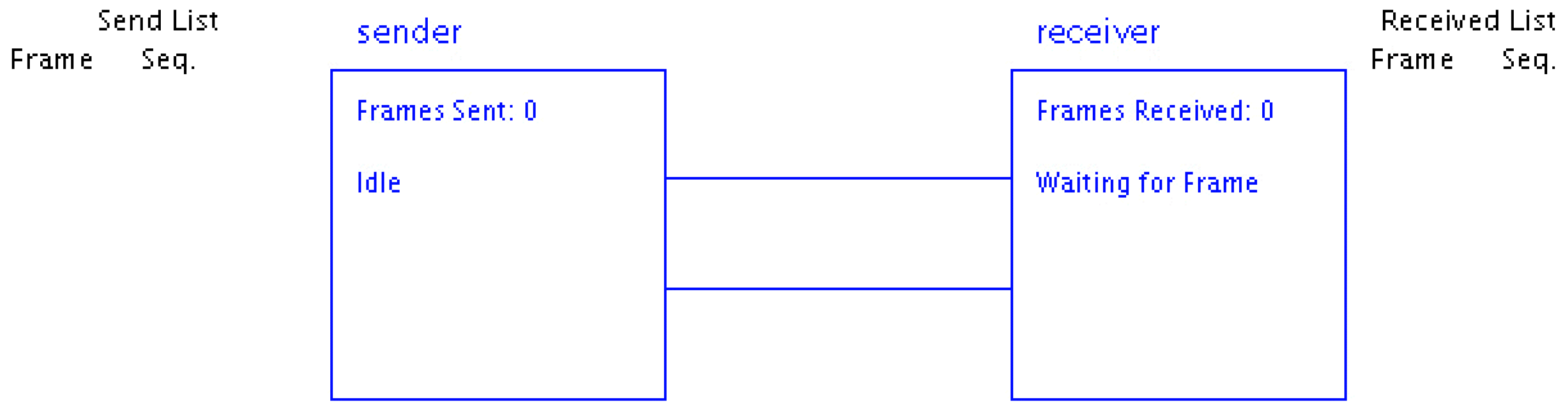
P2: Stop-and-Wait (Error Free)

- Assumes no errors~~, and receiver as fast as sender~~
- Receiver returns a dummy frame (ACK) when ready.
- Hence, only one frame out at any one time, i.e. called stop-and-wait
- Protocol ensures sender can't outpace receiver: we added flow control!

DEMO!

P2: Stop-and-Wait (Error Free) *(illustration)*

- **Protocol 2 assumes that no errors occur but takes into account the limited processing speed and buffer space of the receiver. After sending a frame, the sender waits for an acknowledgement. Since no errors can occur, the acknowledgement does not need to contain any additional information.**
- *Scenario 2: In this scenario, 3 frames are sent as quickly as possible. After each frame is sent an ACK must be received before the next one is sent.*



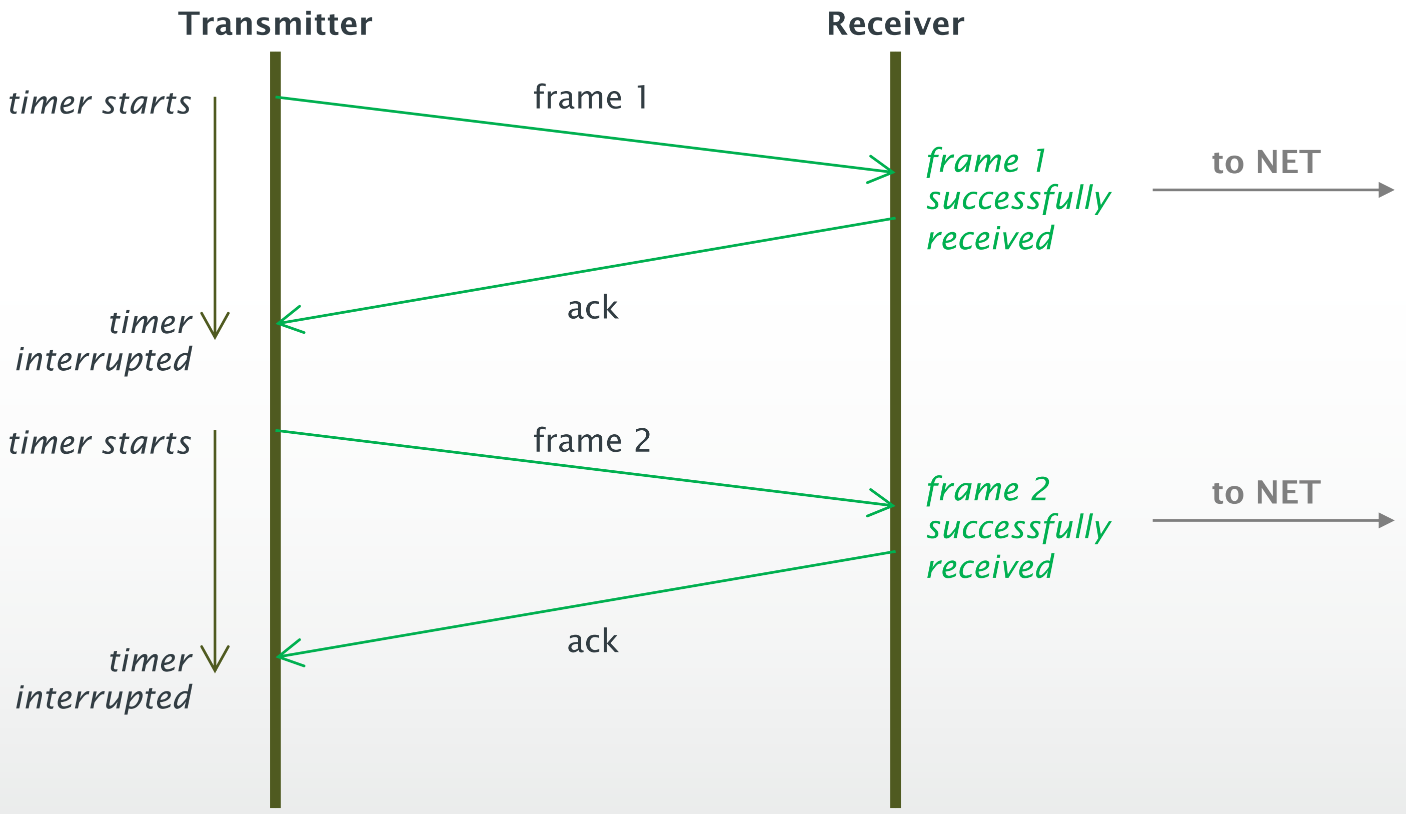
P3: Stop-and-Wait (Noisy Channel)

- If we have a noisy channel, three things could happen
  - The frame is successfully received at the receiver, and the ACK is successfully received at the transmitter. Everything is fine.
  - The frame is successfully received at the receiver, but the ACK is corrupted/lost and hence not received at the transmitter. The system hangs.
  - The frame is corrupted/lost and hence not received at the receiver, and hence the ACK is never transmitted (or received at the transmitter). The system hangs.

- Why not use the dummy ACK packets to determine whether or not the frame needs to be resent?
  - The sender starts a timer whenever it transmits a frame
  - If an ACK is received before the timer expires, the next frame is transmitted
  - If an ACK isn't received when the timer expires, the same frame is retransmitted
- This is known as ARQ (Automatic Repeat reQuest)

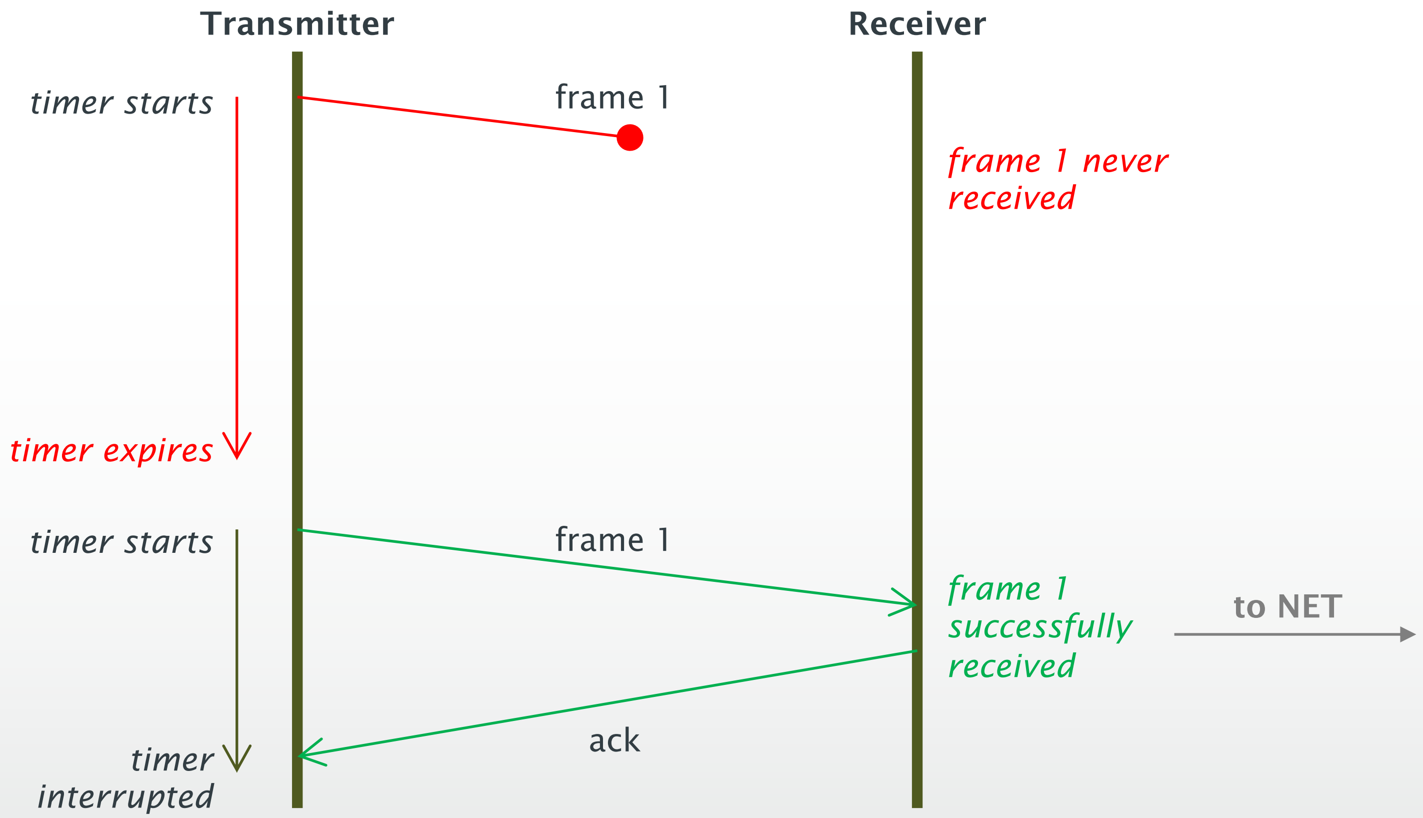
P3: Stop-and-Wait (Noisy Channel)

- Normal operation:



P3: Stop-and-Wait (Noisy Channel)

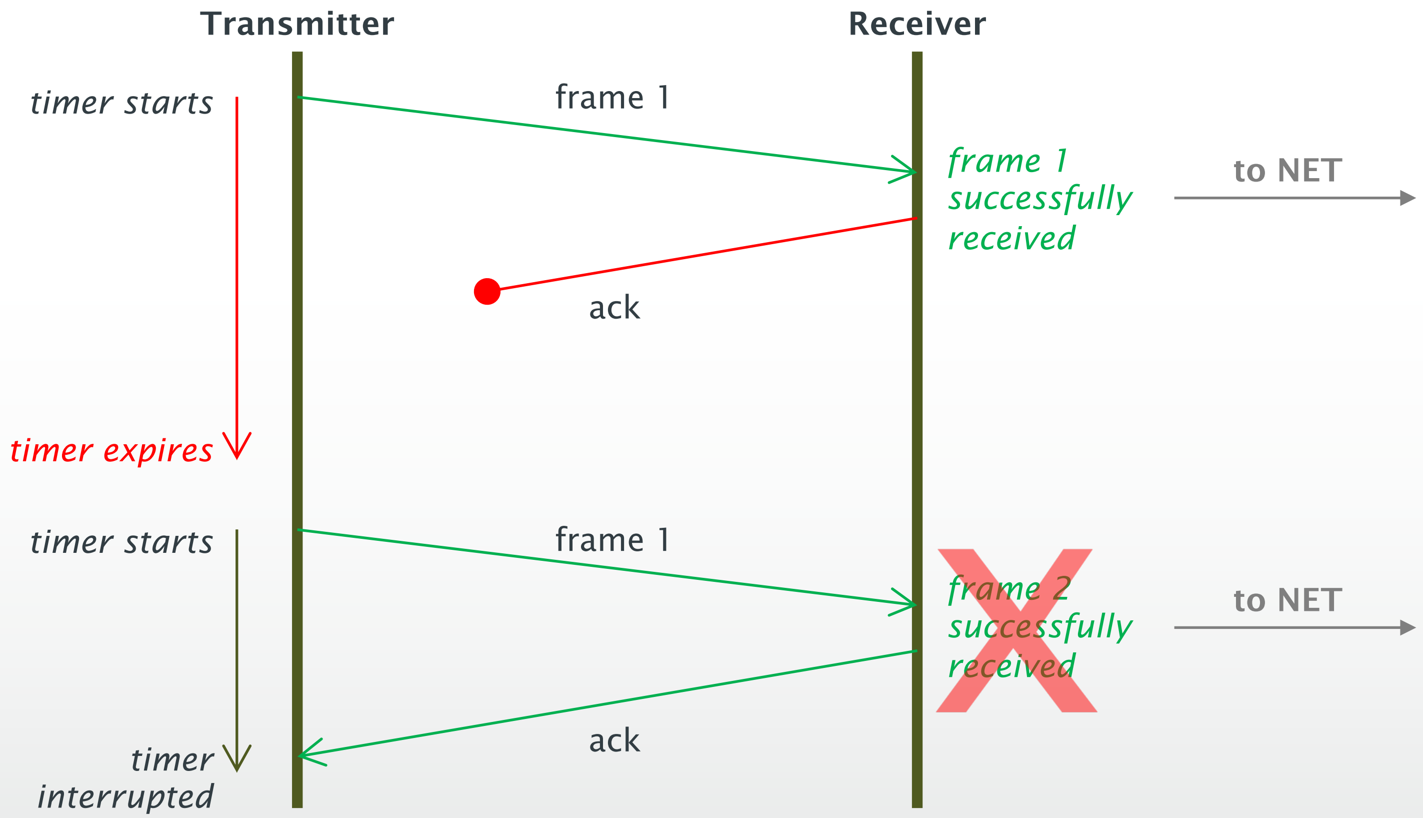
- Lost frame:



- Looks good – *so what's the problem*?

P3: Stop-and-Wait (Noisy Channel)

- Lost acknowledgement:



P3: Stop-and-Wait (Noisy Channel)

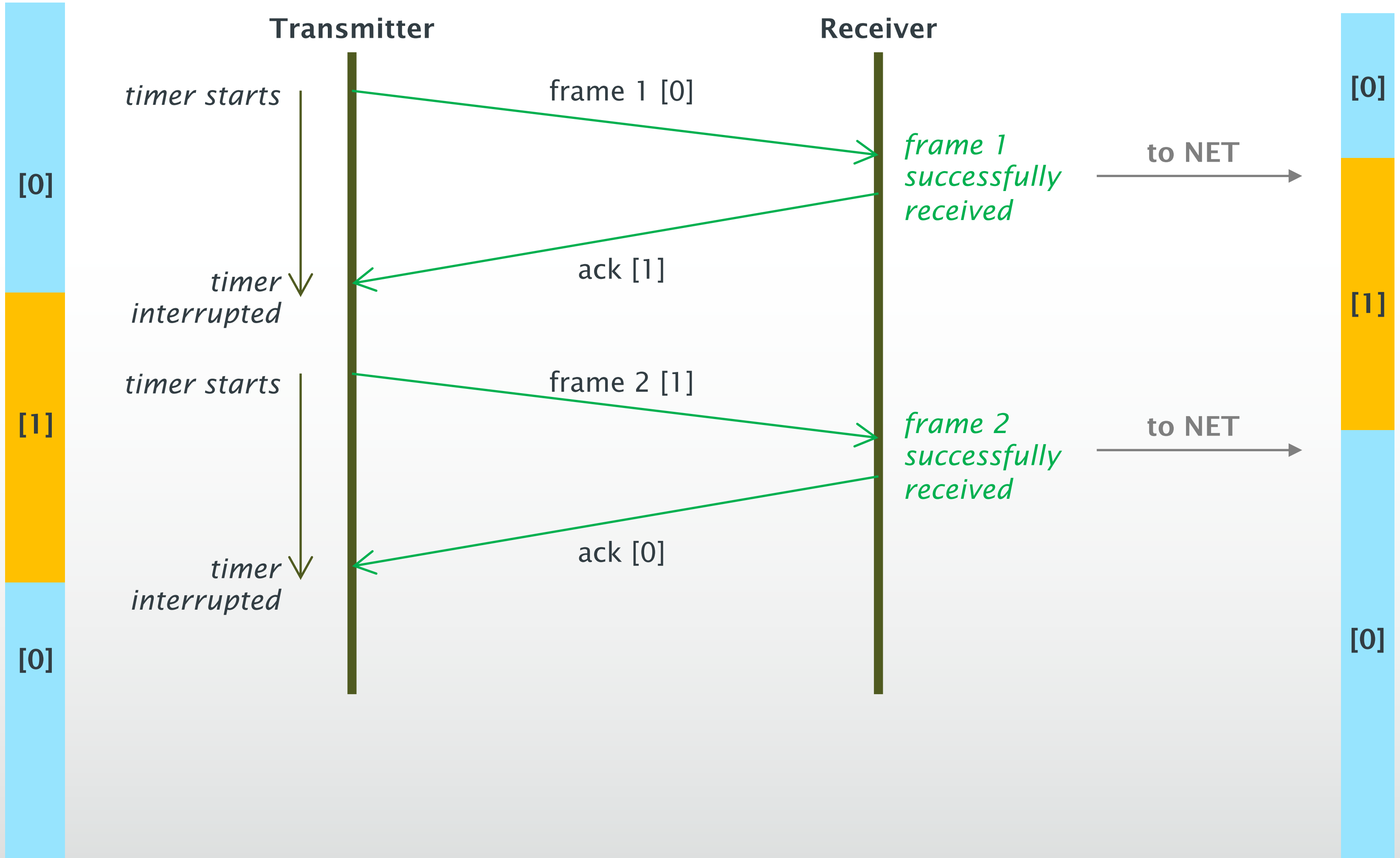
- We therefore need to number frames and acknowledgements
  - Else receiver can't tell retransmission (due to lost ACK/early timer) from new frame
  - For stop-and-wait, 2 numbers (1 bit) are sufficient

- These numbers are *sequence numbers*

- For a frame/transmitter
  - The sequence number indicates the frame it is currently trying to transmit
- For an ACK/receiver
  - The sequence number indicates the frame that the receiver is ready to receive

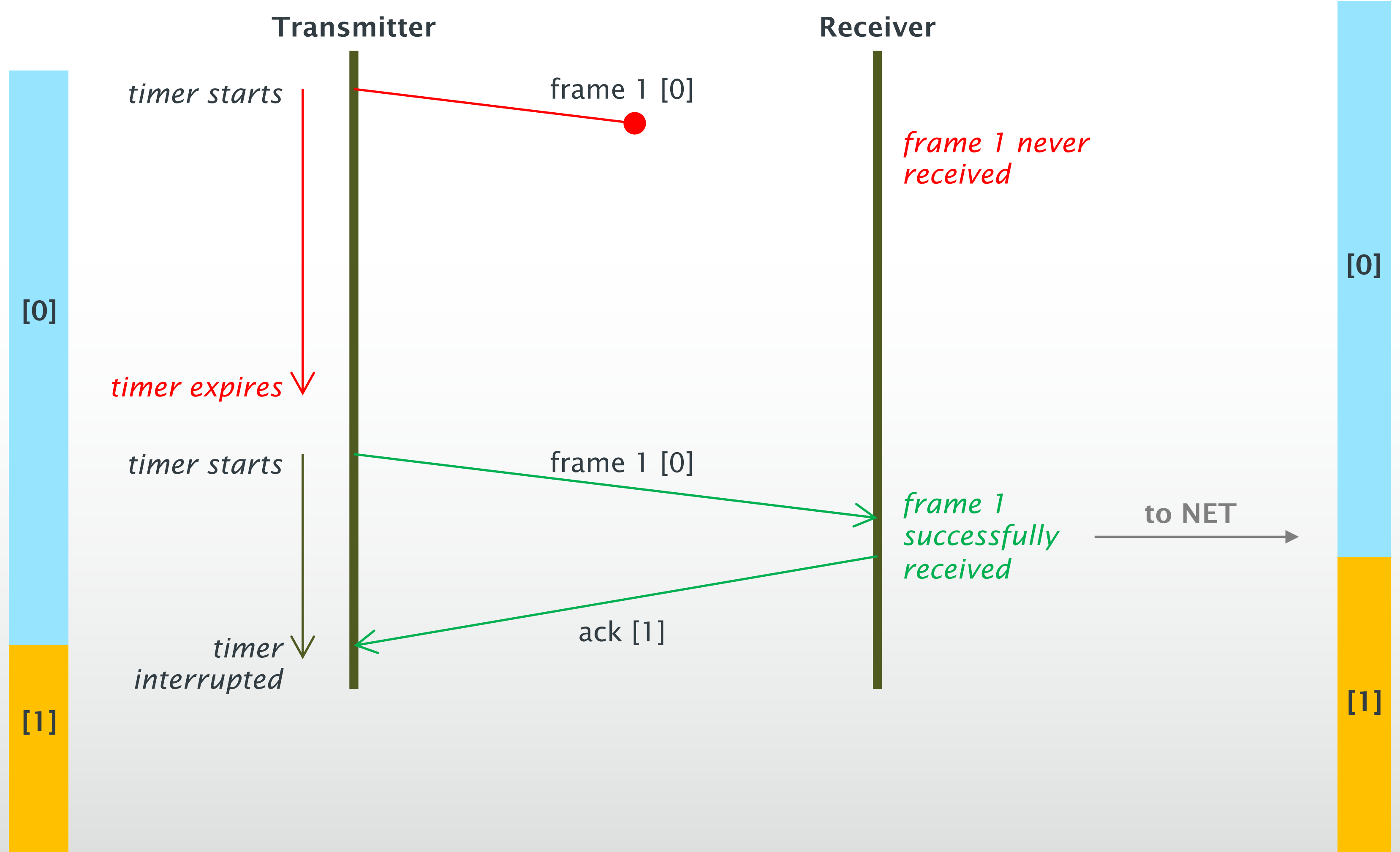
P3: Stop-and-Wait (Noisy Channel)

- With sequence numbers (in square brackets):



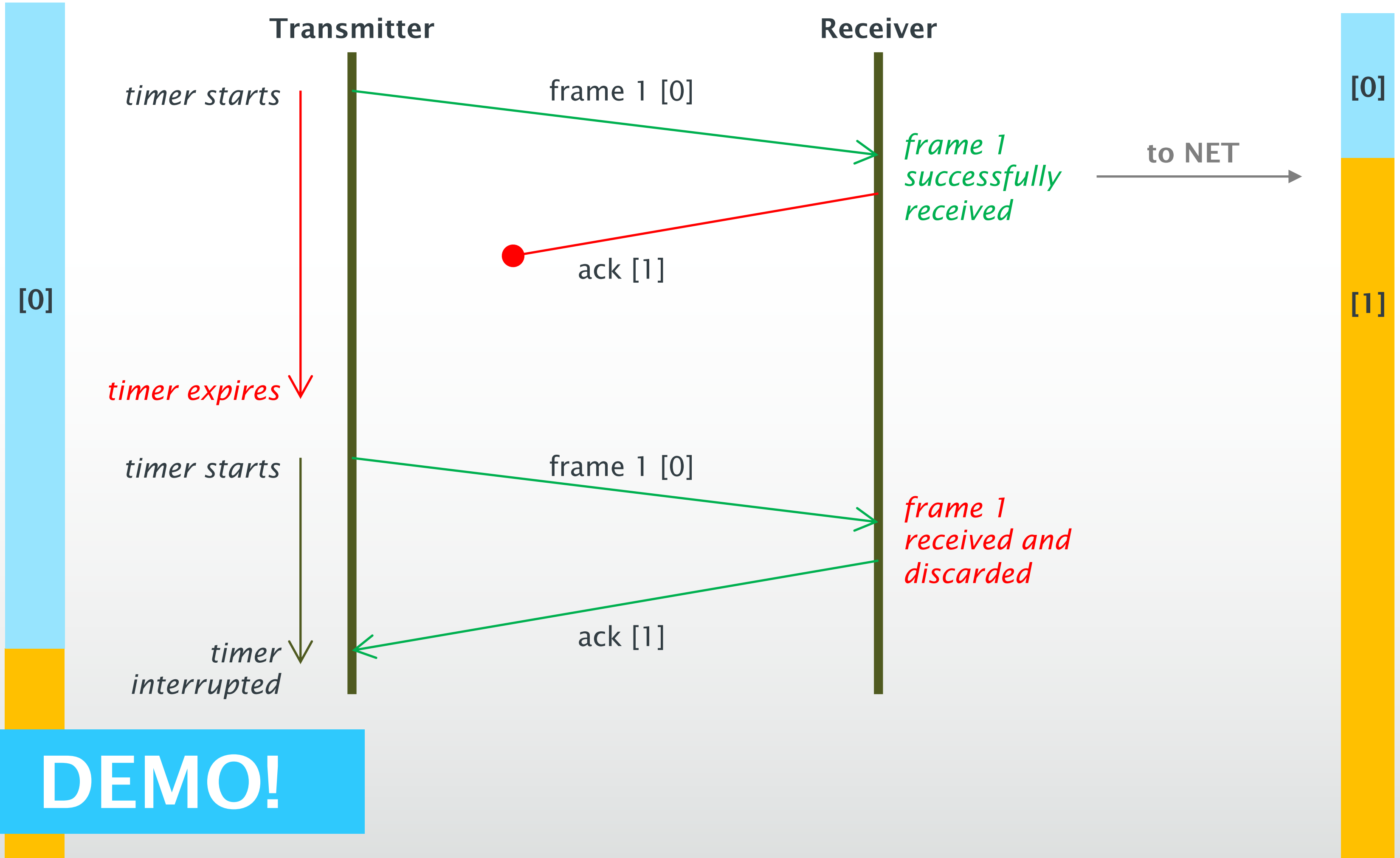
P3: Stop-and-Wait (Noisy Channel)

- With sequence numbers (in square brackets):



P3: Stop-and-Wait (Noisy Channel)

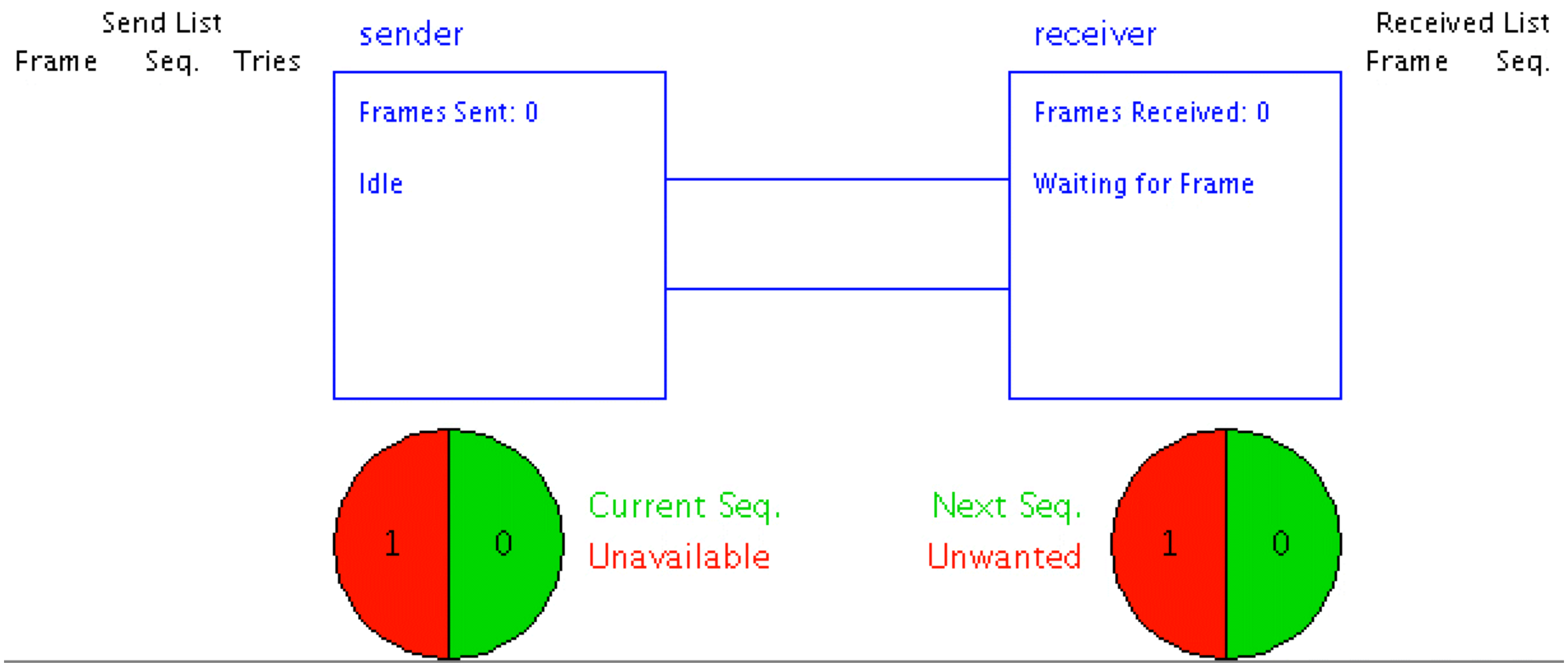
- With sequence numbers (in square brackets):



DEMO!

P3: Stop-and-Wait (Noisy Channel) *(illustration)*

- **Protocol 3 handles errors by waiting for an acknowledgement for each frame sent. If an acknowledgement is not received within a given time the frame is retransmitted. Frames have one-bit sequence numbers (0 and 1) so that the receiver can distinguish new from duplicate frames. This accounts for lost frames or ACKs. ACKs do not have sequence numbers. An ACK acknowledges the last frame that was sent.**
- *Scenario 1: One frame is sent but it is lost. The sender times out and resends the frame. This time it gets through successfully.*



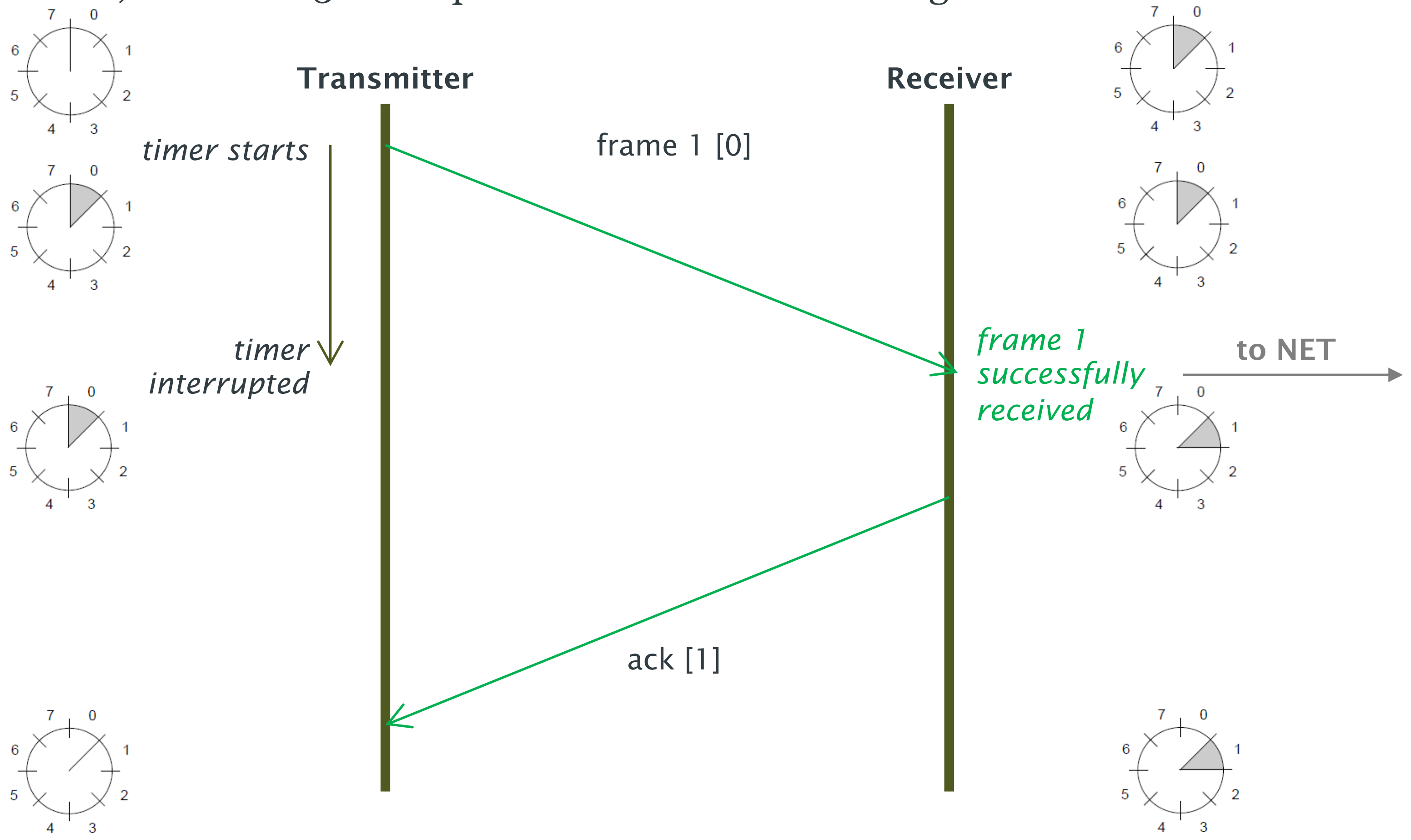
Sliding Window Protocols

- Sender maintains window of frames it can send
  - Needs to buffer them for possible retransmission
  - Window advances with next acknowledgements

- Receiver maintains window of frames it can receive
  - Needs to keep buffer space for arrivals
  - Window advances with in-order arrivals

P4: One-Bit Sliding Window

- The same as Stop-and-Wait ARQ, but with non-binary sequence numbers
  - Here, we have a 3-bit sequence number with a sliding window of size 1



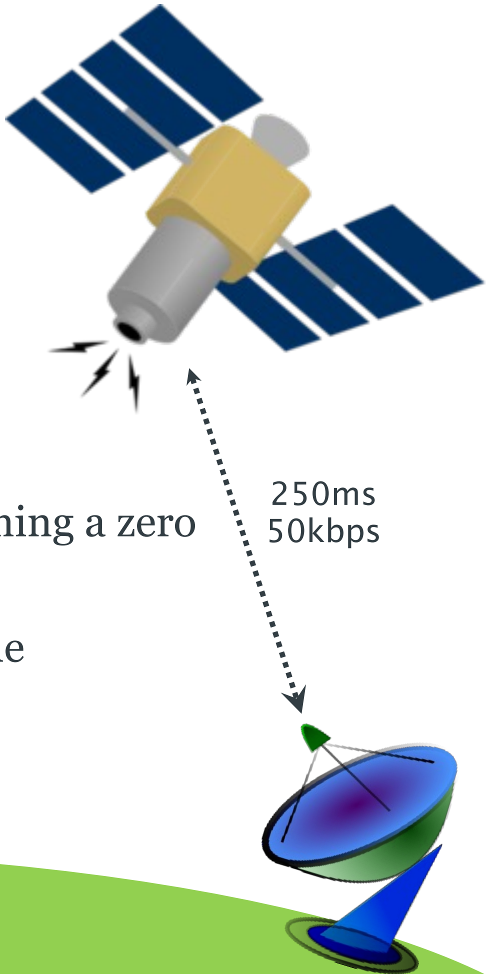
P4: One-Bit Sliding Window

- Transmitter
  - Whenever the NET passes a packet to the DLL, it is given the next highest sequence number and the upper edge of the window is incremented by 1
  - If the window grows to its maximize size (in this case, 1!), the DLL forcibly stops the NET passing it any more packets
  - Whenever an acknowledgement is received, the lower edge is advanced by 1
- Receiver
  - Any frame received that falls within the current window is put in the frame buffer
  - If a frame corresponding to the lower edge of the window is received, it is passed to the NET and the window rotated by 1
  - If a frame with a sequence number outside the window is received, it is discarded

Sliding Window Protocols

- So if it's the same as Stop-and-Wait ARQ, why bother with the additional complexity?

- Suppose we use this 1-bit sliding window protocol to transmit 1000-bit frames.
  - t=0ms: sender starts transmitting first frame
  - t=20ms: first frame sent
  - t=270ms: first frame received by satellite
  - t=520ms: sender received ACK for first frame (assuming a zero length acknowledgement)
  - t=520ms, sender starts transmitting the second frame
- Sender was 'blocked' for 500/520ms (i.e. 96%)
  - Only 4% of the available bandwidth was used



Sliding Window Protocols

- To resolve this, instead of waiting for one frame to be sent-received-acknowledged before sending the next…
- …we want to be able to queue up lots of frames being sent and awaiting acknowledgements
  - Effectively, pipelining!

- Bandwidth-Delay = bandwidth [bps] * one-way transmit time [s]
- BD = Bandwidth-Delay [b] / frame size [b]
- An appropriate sliding-window size can be obtained by using:

$$w = 2\text{BD} + 1$$

  - In the previous example, we would like the transmitter to have just finished sending frame 26 when the first acknowledgement comes back.

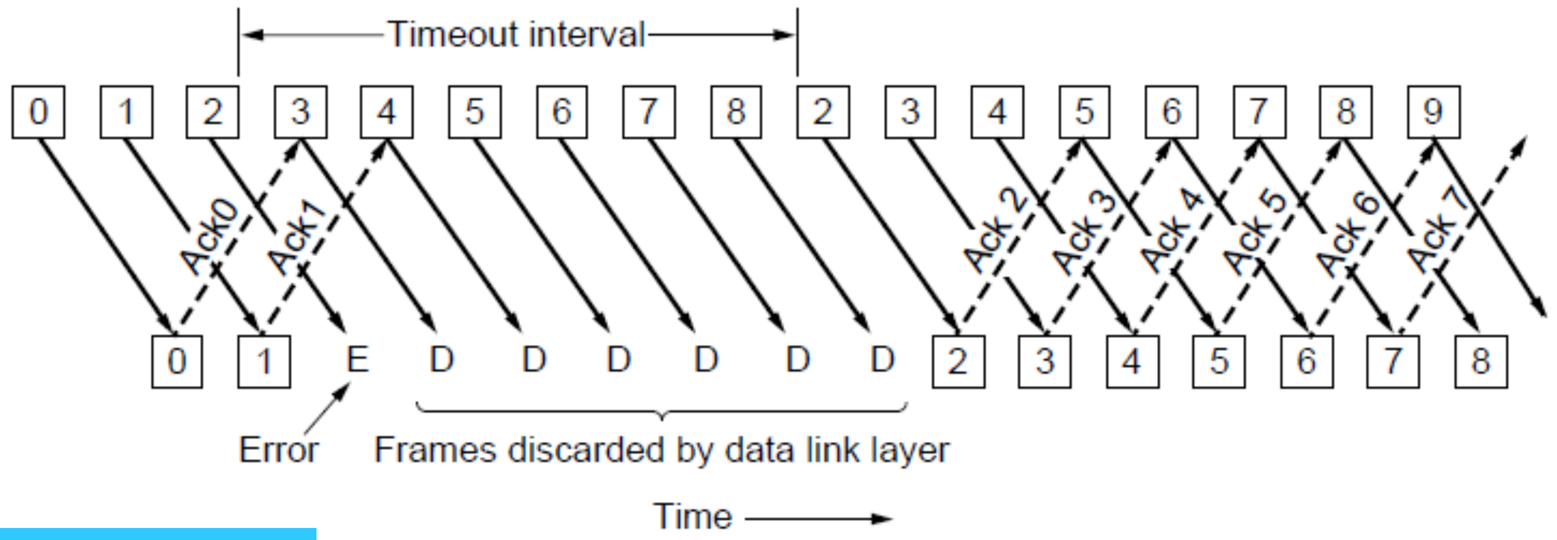
Sliding Window Protocols

- This leads to different ways to handling errors:
  - Go-back-N
  - Selective Repeat

- Trade bandwidth for memory/buffers

P5: Go-Back-N

- Receiver only accepts/ACKS frames that arrive in order:
  - A receive window of size *N*
  - Discards frames that follow a missing/errored frame
  - Sender times out and resends all outstanding frames

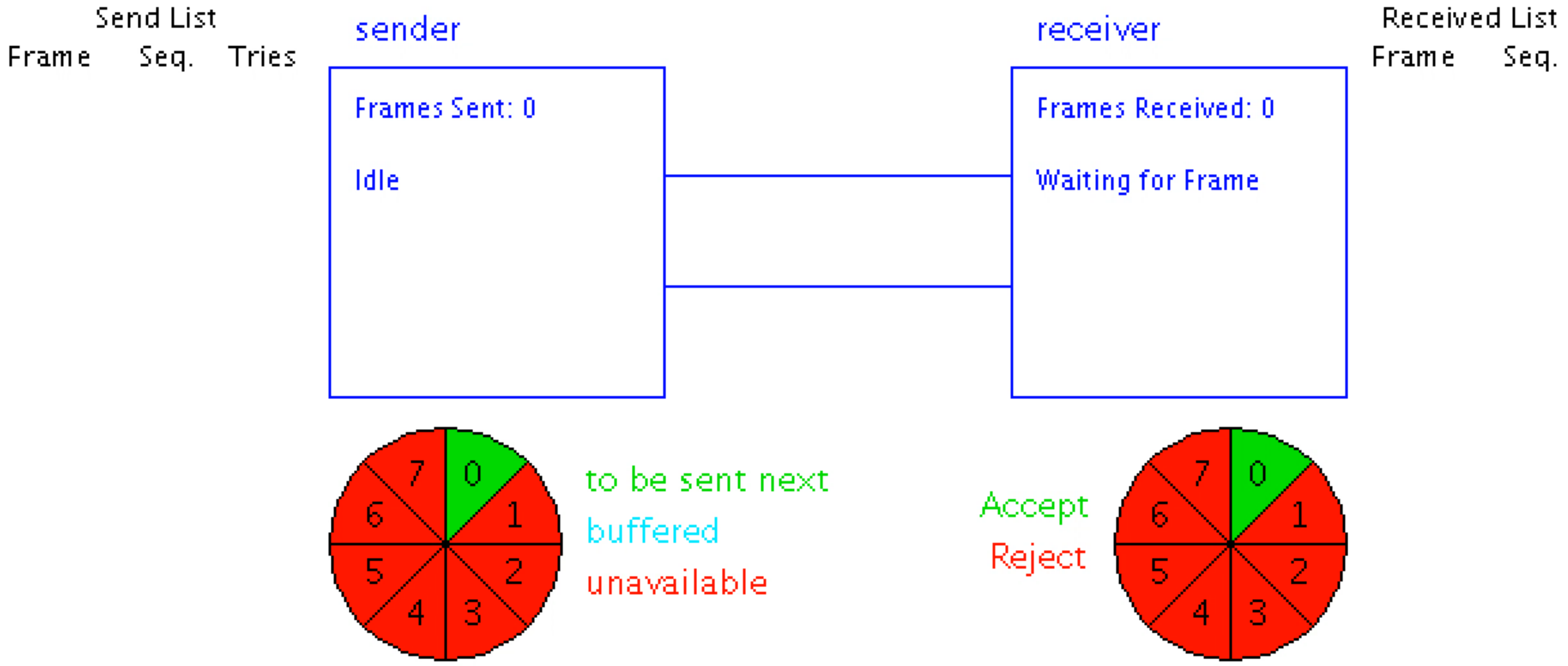


DEMO!

CN5E by Tanenbaum & Wetherall, © Pearson Education-Prentice Hall and D. Wetherall, 2011

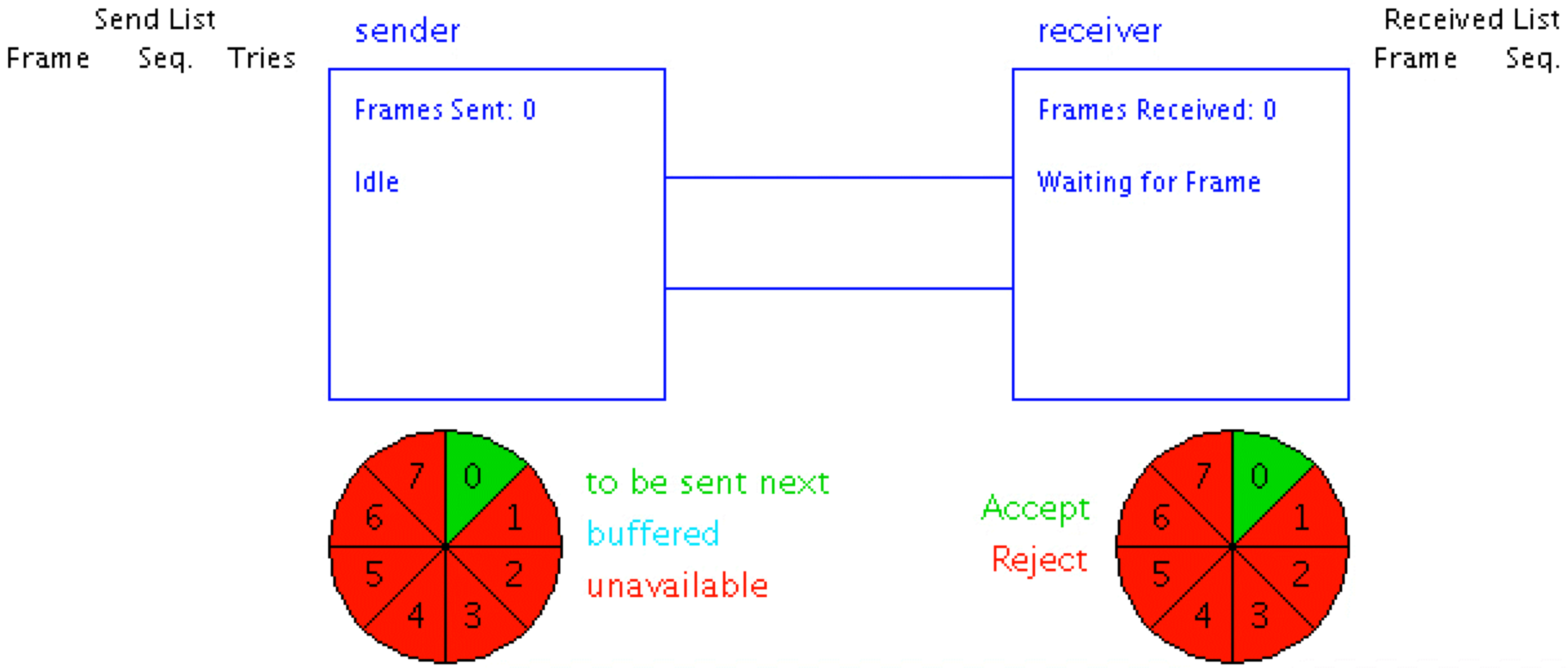
P5: Go-Back-N *(illustration)*

- **Protocol 5 allows pipelining. In these examples, the maximum sequence number is 7 which allows for 8 sequence numbers and 7 outstanding frames. 7 frames can be sent before the sender is blocked. The receiver has a window size of 1, meaning that it cannot accept frames out of order. When it receives a frame, the receiver sends an ACK for the last frame it has accepted. When the sender receives an ACK for frames it has buffered, it frees those buffers and additional frames can be sent.**
- *Scenario 1: Protocol 5 scenario 1 sends 7 frames. It is assumed that the sender has only 7 frames to send. Frame 2 is lost and frames 2, 3, 4, 5, and 6 have to be resent when frame 2 times out. The ACK for frame 4 is lost but it doesn't matter since the ACK for 5 is received before frame 4 times out.*



P5: Go-Back-N *(illustration)*

- **Protocol 5 allows pipelining. In these examples, the maximum sequence number is 7 which allows for 8 sequence numbers and 7 outstanding frames. 7 frames can be sent before the sender is blocked. The receiver has a window size of 1, meaning that it cannot accept frames out of order. When it receives a frame, the receiver sends an ACK for the last frame it has accepted. When the sender receives an ACK for frames it has buffered, it frees those buffers and additional frames can be sent.**
- *Scenario 2: This is identical to Scenario 1, except that the sender has 16 frames to send instead of just 7. As it receives ACKs for frames 0 and 1 it send additional frames. When it times out for frame 2 it resends frames 2-8.*

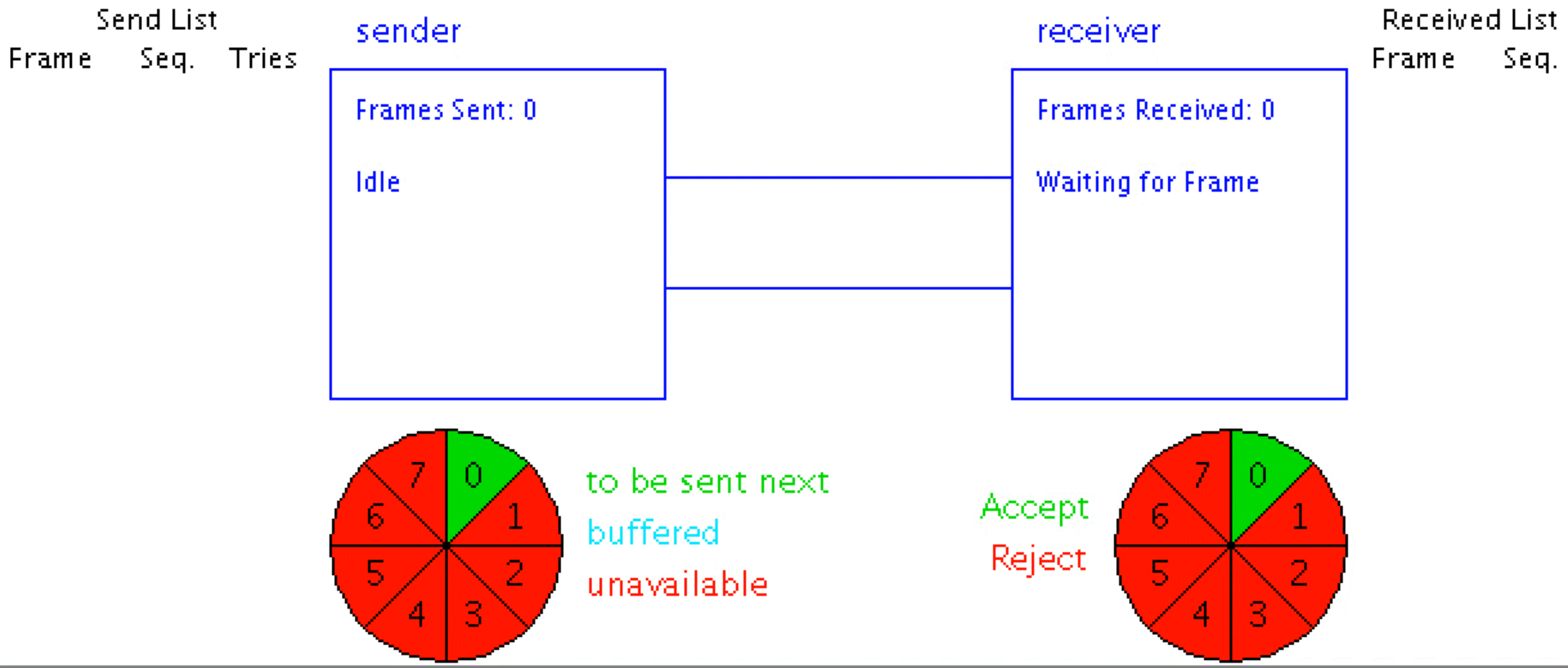


Acknowledgments

- Acknowledgements
  - Positive acknowledgement (ACK):
    - sent by a receiver, to the transmitter, after successful receipt of a frame
  - Negative acknowledgement (NAK):
    - sent by a receiver, to the transmitter, after unsuccessful receipt of a frame

P5: Go-Back-N *(illustration)*

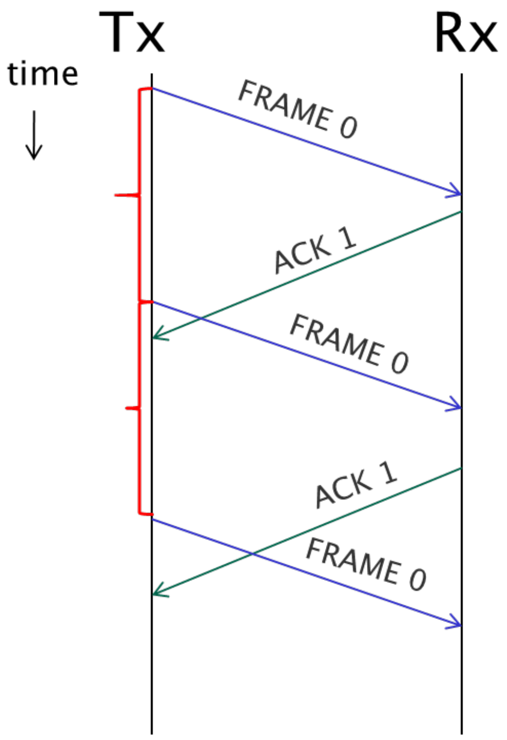
- **Protocol 5 allows pipelining. In these examples, the maximum sequence number is 7 which allows for 8 sequence numbers and 7 outstanding frames. 7 frames can be sent before the sender is blocked. The receiver has a window size of 1, meaning that it cannot accept frames out of order. When it receives a frame, the receiver sends an ACK for the last frame it has accepted. When the sender receives an ACK for frames it has buffered, it frees those buffers and additional frames can be sent.**
- *Scenario 4: Protocol 5 scenario 4 is identical to scenario 3 except that the protocol is modified to treat the first unexpected ACK for a frame as a NAK. In this case the lost frames are resent before the timeout occurs.*



Design consideration: Timeout

- **Too short**: inefficient as frames are unnecessarily retransmitted
- **Too long**: inefficient when frame lost
- We need to find the Goldilocks solution!

  (Neither too short nor too long)



P5: Go-Back-N

- Go-back-N
  - Sender has to buffer all frame, in case it has to ‘go-back’
  - Receiver doesn’t buffer frames (just waits for the one that it is expecting)
  - Good if errors are rare. If common, wastes a lot of bandwidth through retransmission

P6: Selective Repeat

- Receiver accepts frames anywhere in receive window
  - Works the same as Go-back-N, If an erroneous frame is received, it is discarded.
  - However, any good frames received after it are buffered (i.e. receiver window >1)
  - When the sender times out, only the oldest unacknowledged frame is retransmitted

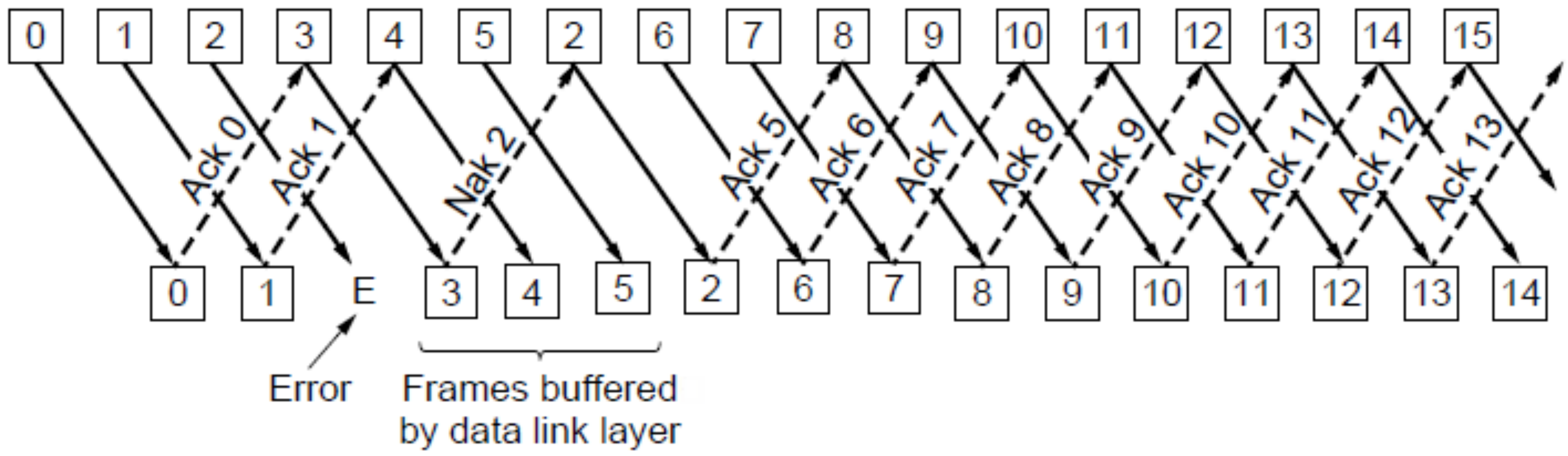
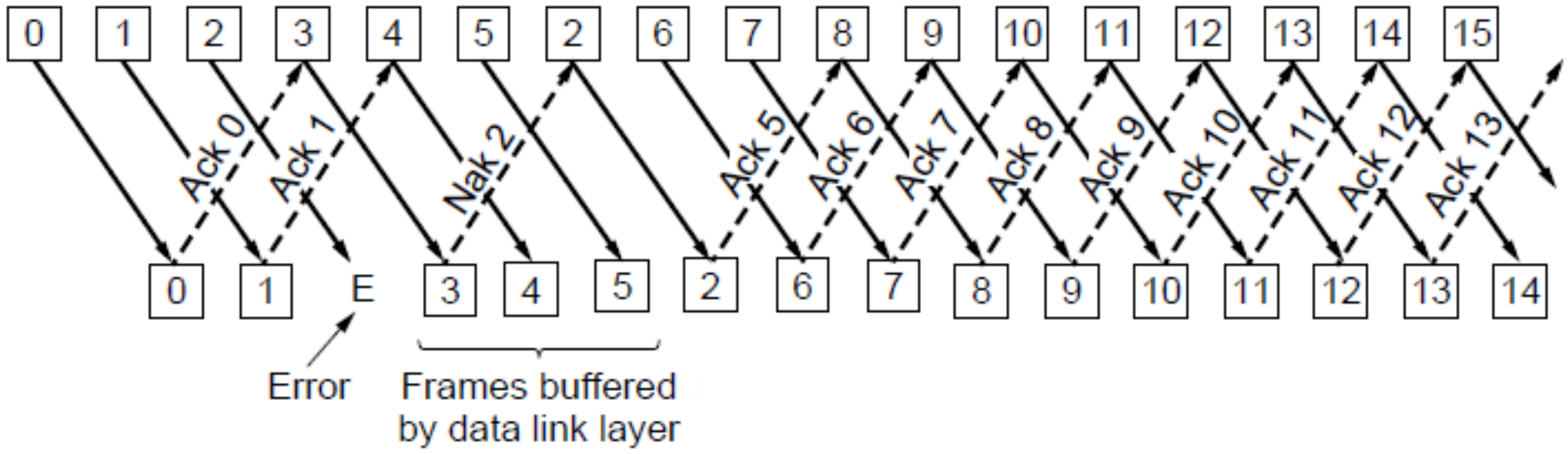


Image: CN5E by Tanenbaum & Wetherall, © Pearson Education-Prentice Hall and D. Wetherall, 2011

P6: Selective Repeat

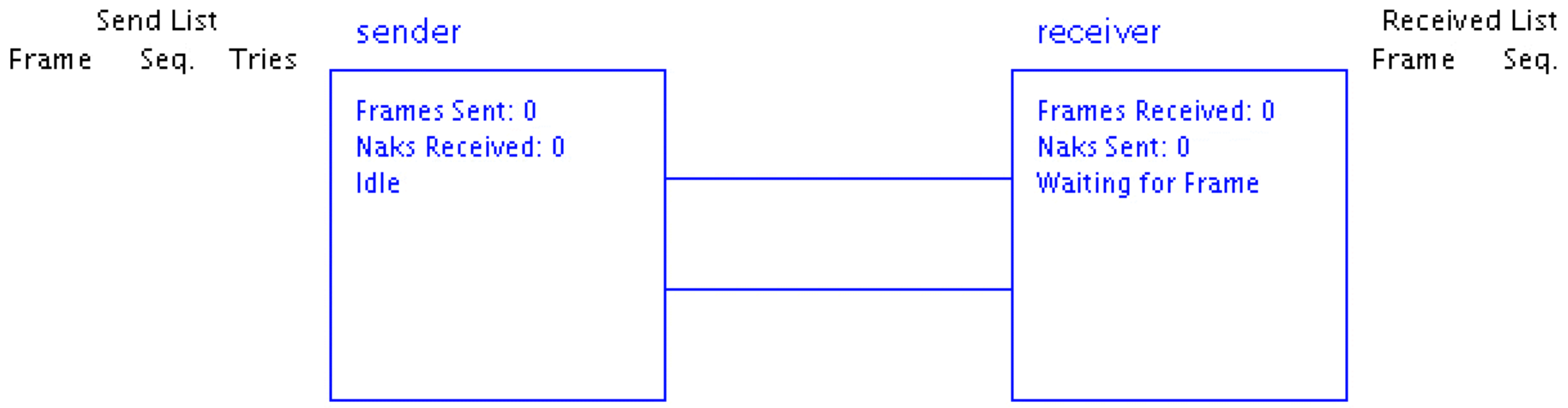
- Receiver accepts frames anywhere in receive window
  - Can improve efficiency by sending a NAK if an error is detected, which causes the sender to retransmit a missing frame before a timeout
  - Cumulative ACK indicates highest in-order frame
  - Each sequence number in the window has a receive buffer associated with it, and a bit depicting whether that buffer is full or empty



DEMO!

P6: Selective Repeat *(illustration)*

- **Protocol 6 allows pipelining and non-sequential receive. It also supports negative acknowledgements. In the examples the maximum sequence number is 7 which allows for up to 4 outstanding frames. The receiver's window size is 4. The negative acknowledgements allow for a longer timeout without significantly sacrificing performance.**
- *Scenario 1: Protocol 6 scenario 1 is similar to Protocol 5, scenario 3. Six frames are to be sent. Frame 2 gets lost causing a NAK to be sent. When the sender receives the NAK, only frame 2 is send again. The timeout is set very long and does not affect the protocol in this case.*



P6: Selective Repeat

- Go-back-N
  - Sender has to buffer all frame, in case it has to 'go-back'
  - Receiver doesn't buffer frames (just waits for the one that it is expecting)
  - Good if errors are rare. If common, wastes a lot of bandwidth through retransmission

- Selective Repeat
  - More complex than Go-back-N due to receiver buffers and multiple sender timers
  - More efficient use of link bandwidth as only lost frames are resent
  - Good if errors are common. Requires a lot more memory/buffers

Duplex Data Transfer

- Although there's been bi-direction communication, data transfer has only really been one way (left –> right)
  - Anything going from left -> right was a frame
  - Anything going from right -> left was an ACK
- We can extend this by communicating frames and ACKs in both directions
  - And using a field in the frame header to indicate what type it is
- Sending an ACK is also reasonably inefficient
  - If there is a frame to send, we can piggyback an ACK into a frame header

Electronics and
Computer Science

UNIVERSITY OF Southampton

ELEC3222 18/19 Exam Question

You wish to send data to a geostationary satellite 35786 km directly above you. The full-duplex link offers a data rate of 2.2 kbps, transmits 24-byte frames, and the RF signals travel at an average speed of 2.6×10^8 ms^{-1}. The satellite takes 50 ms to process each received frame before acknowledging.

1. Assuming that a Sliding Window protocol is used, **calculate** the window size required which maximises channel efficiency.

2. **Explain**, using a diagram, why sequence numbers are needed in a Stop-and-Wait ARQ protocol.

Questions?