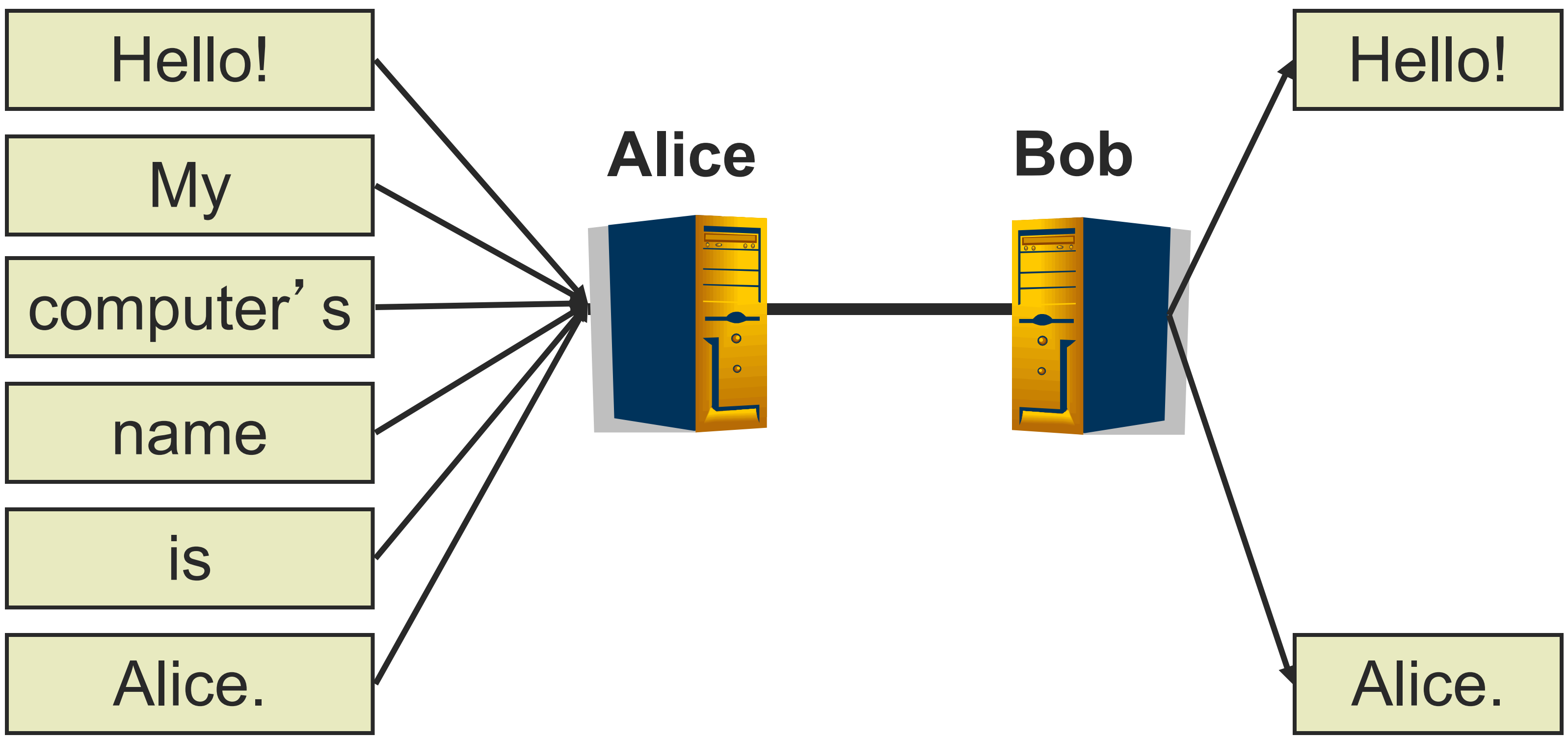
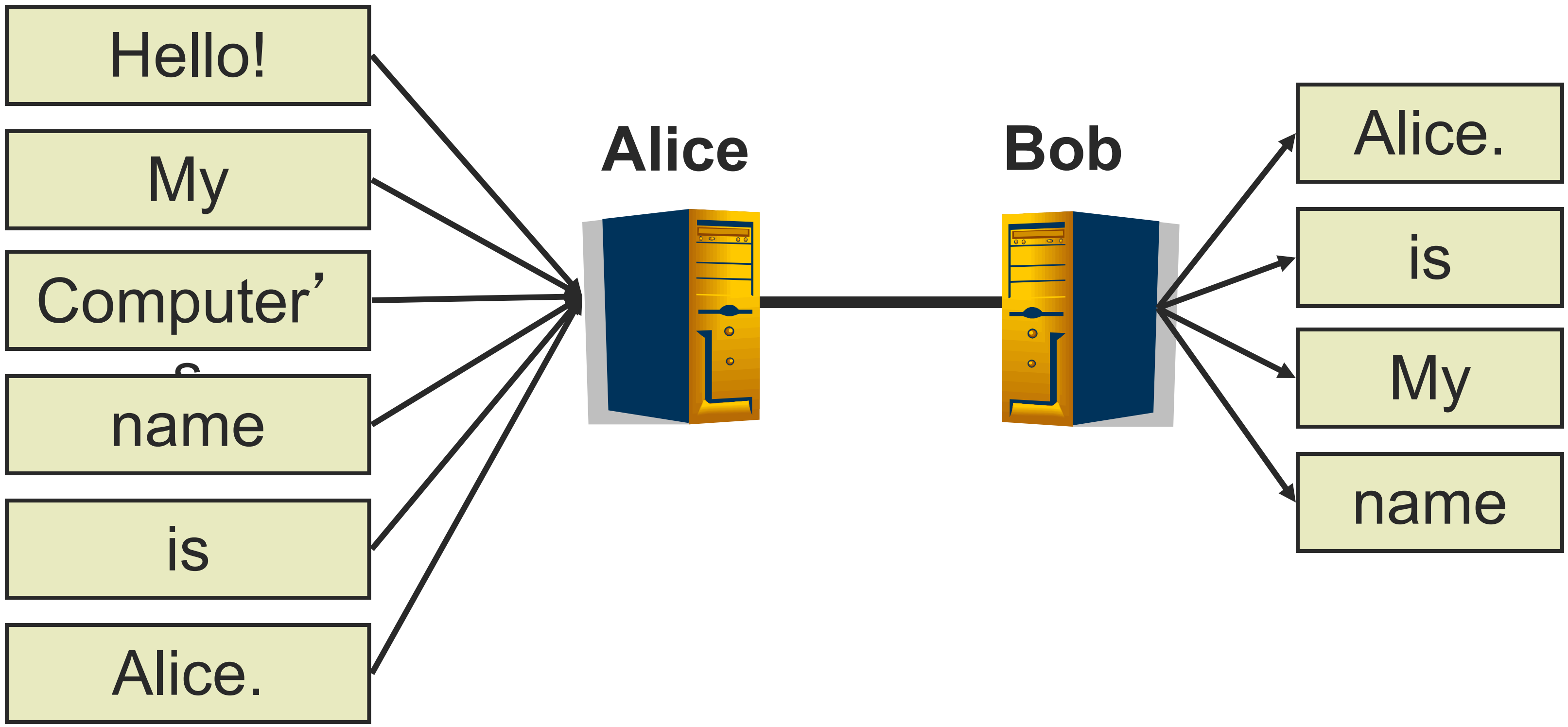
Reliable Transmission

Reliable Transmission



Reliable Transmission



Reliable Transmission

- Suppose error protection identifies valid and invalid packets
  - How?
- Can we make the channel appear reliable?
  - Insure packet delivery
  - Maintain packet order
  - Provide reliability at full link capacity

Reliable Transmission Outline

- Fundamentals of Automatic Repeat reQuest (ARQ) algorithms
  - A family of algorithms that provide reliability through retransmission
- ARQ algorithms (simple to complex)
  - stop-and-wait
  - concurrent logical channels
  - sliding window
    - go-back-n
    - selective repeat
- Alternative: forward error correction (FEC)

Terminology

- Acknowledgement (ACK)
  - Receiver tells the sender when a frame is received
    - Selective acknowledgement (SACK)
      - Specifies set of frames received
    - Cumulative acknowledgement (ACK)
      - Have received specified frame and all previous
    - Negative acknowledgement (NAK)
      - Receiver refuses to accept frame now, *e.g.*, when out of buffer space

Terminology

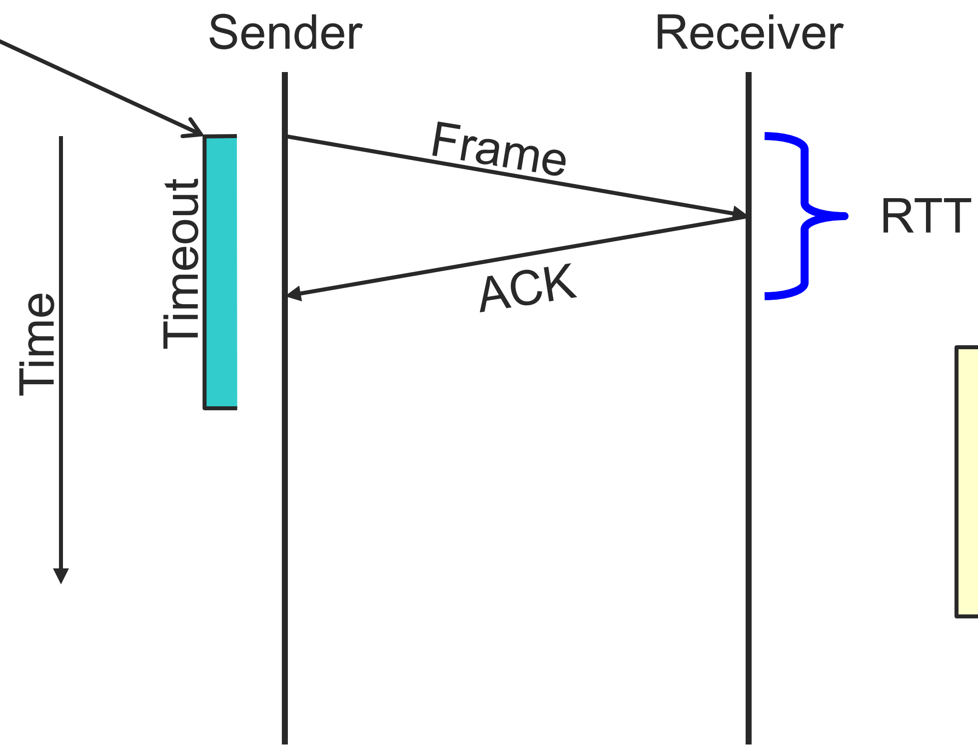
- Timeout (TO)
  - Sender decides the frame (or ACK) was lost
  - Sender can try again

Stop-and-Wait

- Basic idea
  1. Send a frame
  2. Wait for an ACK or TO
  3. If TO, go to 1
  4. If ACK, get new frame, go to 1

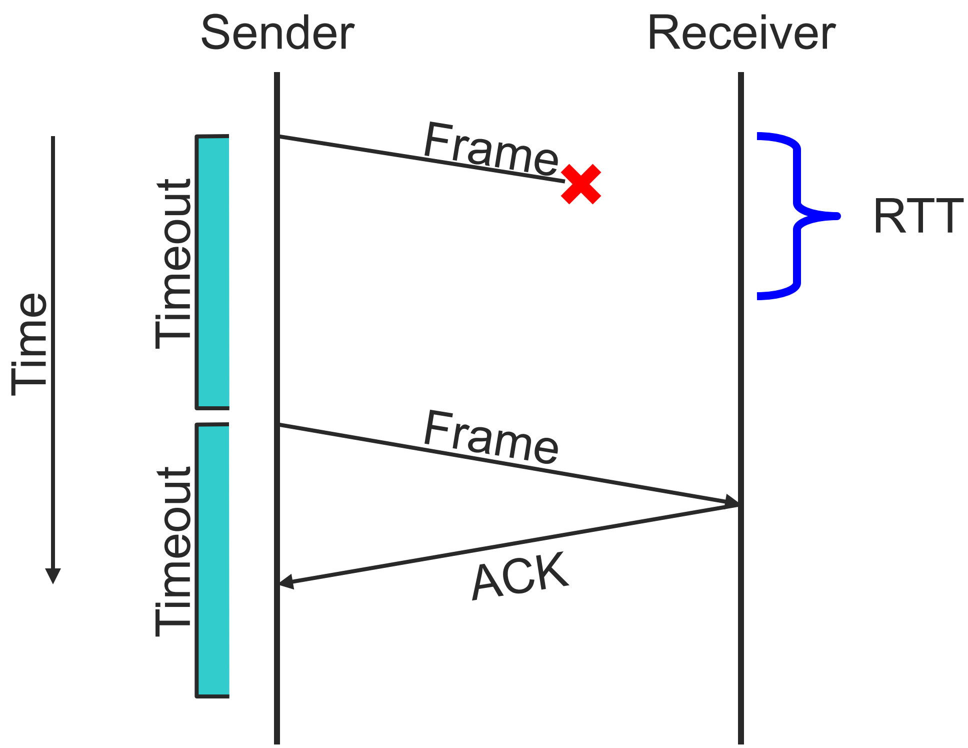
Stop-and-Wait: Success

How long should the timeout be?

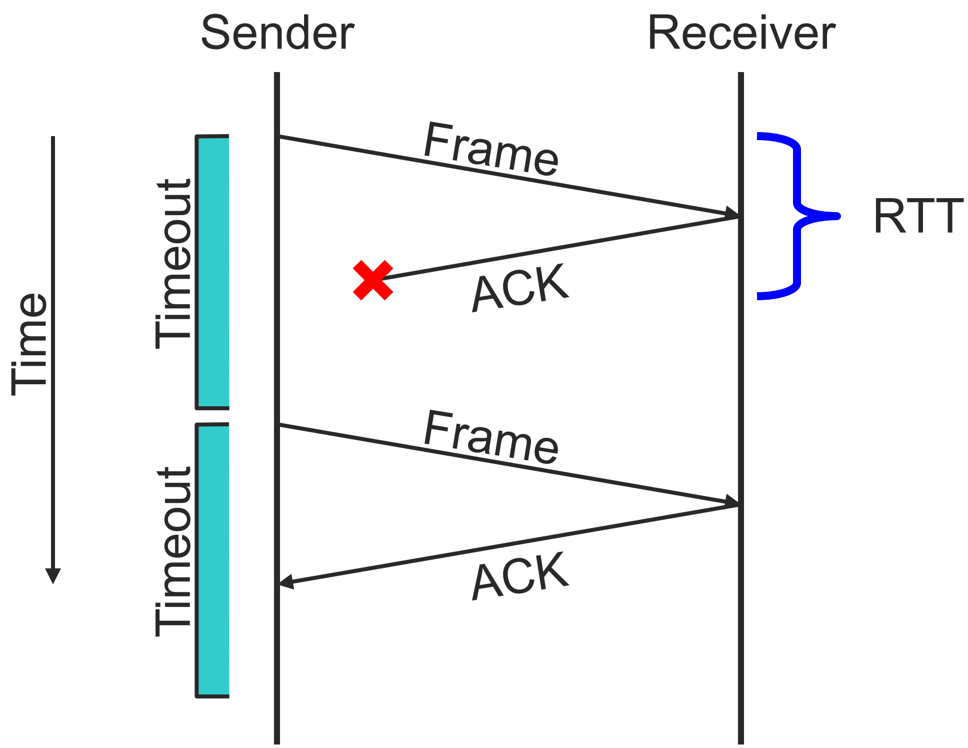


What can go wrong?
How will it affect our protocol?

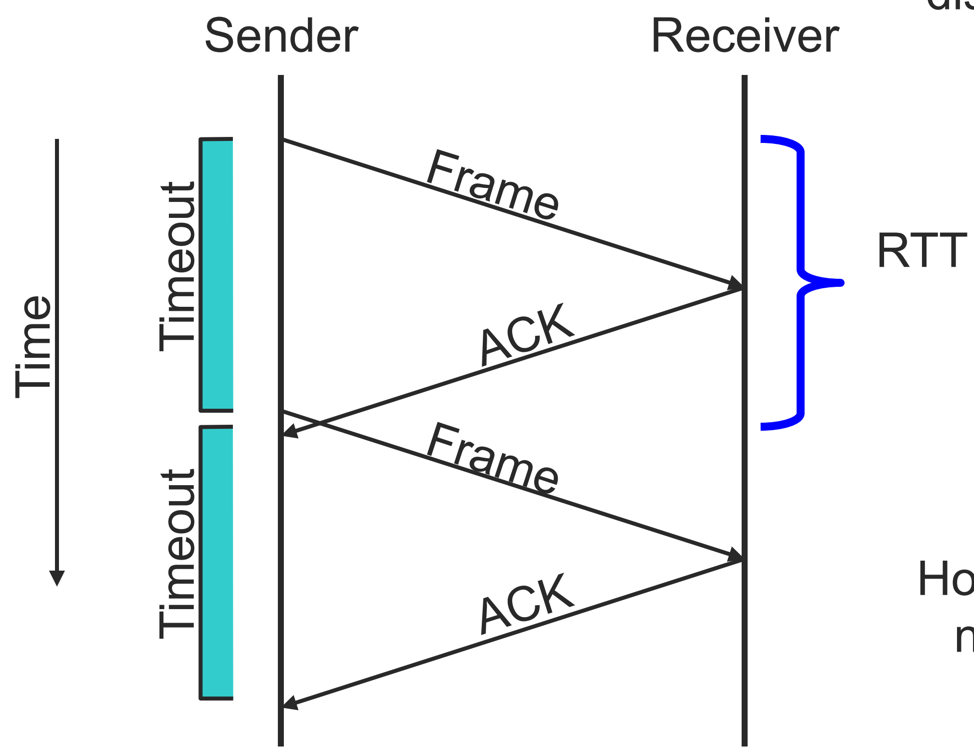
Stop-and-Wait: Lost Frame



Stop-and-Wait: Lost ACK



Stop-and-Wait: Delayed Frame



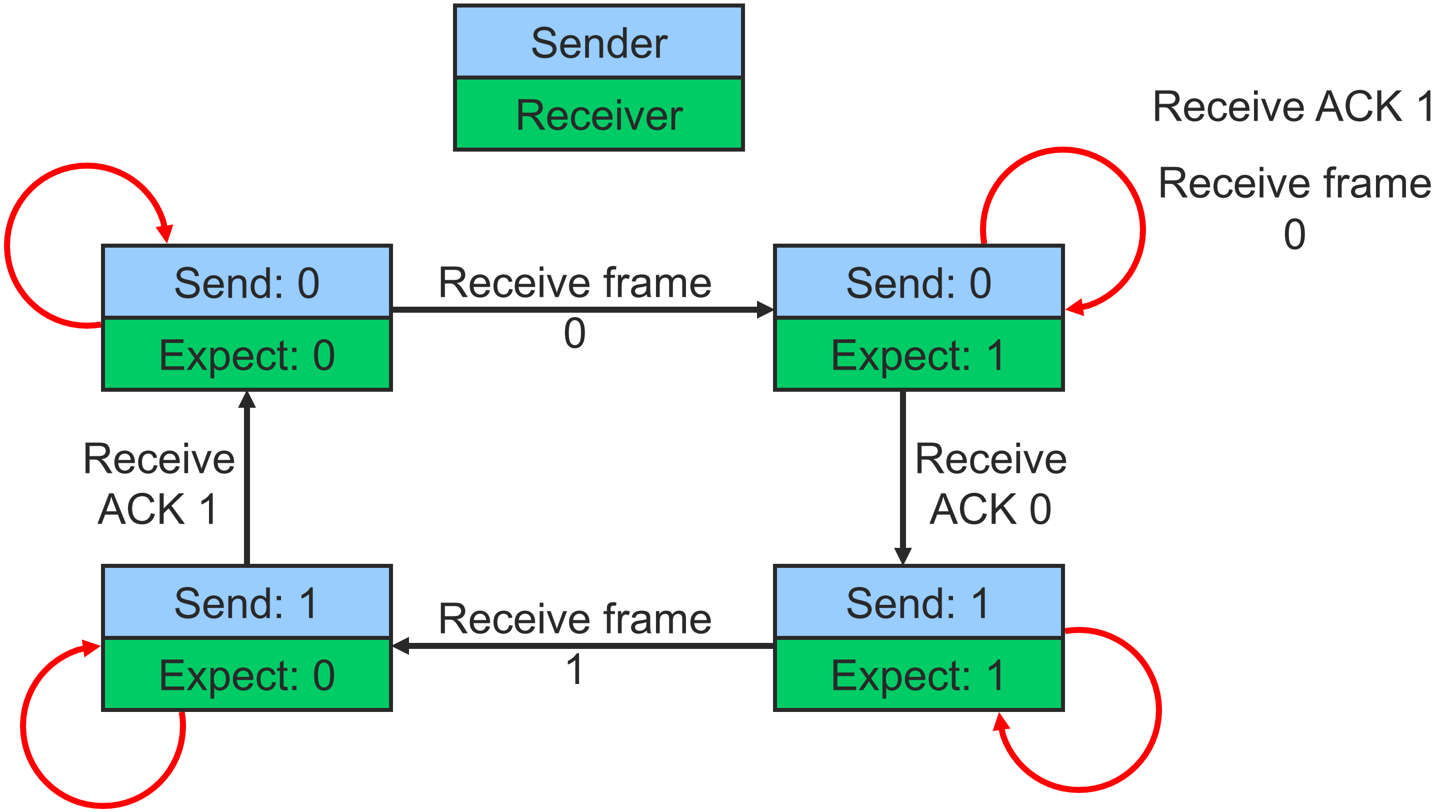
How can receiver distinguish between two frames?

How many bits do you need for sequence numbers?

Stop-and-Wait

- Goal
  - Guaranteed, at-most-once delivery
- Protocol Challenges
  - Dropped frame/ACK
  - Duplicate frame/ACK
- Requirements
  - 1-bit sequence numbers (if physical network maintains order)
    - sender tracks frame ID to send
    - receiver tracks next frame ID expected

Stop-and-Wait State Diagram



Stop-and-Wait

- We have achieved
  - Frames delivered reliably and in order
  - Is that enough?
- Problem
  - Only allows one outstanding frame
    - Does not keep the pipe full
  - Example
    - 100ms RTT
    - One frame per RTT = 1KB
    - 1024x8x10 = 81920 kbps
    - Regardless of link bandwidth!

Concurrent Logical Channels

- Used in ARPANET IMP-IMP protocol
- Idea
  - Multiplex logical channels over a physical link
    - Include channel ID in header
  - Use stop-and-wait for each channel
- Result
  - Each channel is limited to stop-and-wait bandwidth
  - Aggregate bandwidth uses full physical channel
  - Supports multiple communicating processes
  - Can use more than one channel per process

Concurrent Logical Channels

- Problem
  - Bandwidth
    - Use of a single channel per process may waste BW
  - Ordering
    - Use of multiple channel per process does not maintain packet ordering across channels!
    - If application has *n* channels, and one needs a retransmission, it will always be one packet behind the other channels

ARQ: Where are We?

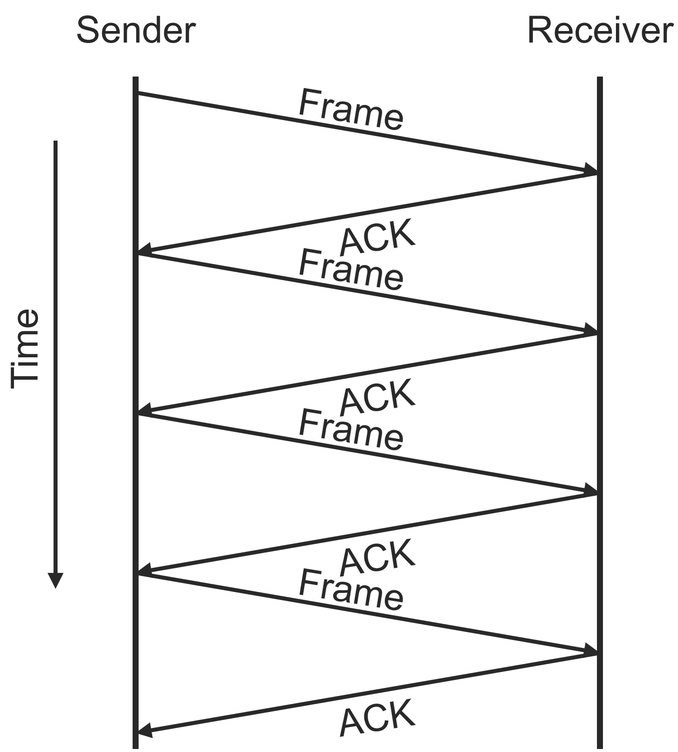
- Goals for reliable transmission
  - Make channel appear reliable
  - Maintain packet order (usually)
  - Impose low overhead/allow full use of link
- Stop-and-Wait
  - Provides reliable in-order delivery
  - Sacrifices performance
- Concurrent Logical Channels
  - Provides reliable delivery at full link bandwidth
  - Sacrifices packet ordering
- Sliding Window Protocol
  - Achieves all three!

Sliding Window Protocol

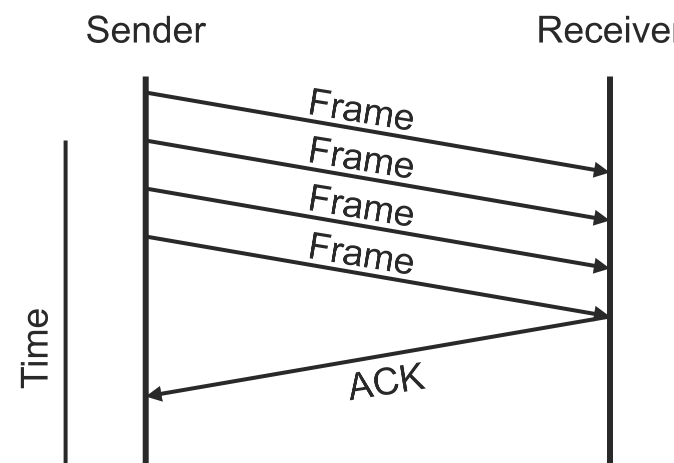
- Most important and general ARQ algorithm
- Used by TCP
- Outline
  - Concepts
  - Terminology (from P&D)
  - Details
  - Code example
  - Proof of eventual in-order delivery
  - Classification scheme
    - (go-back-n, selective repeat)

Keeping the Pipe Full

Stop-and-Wait



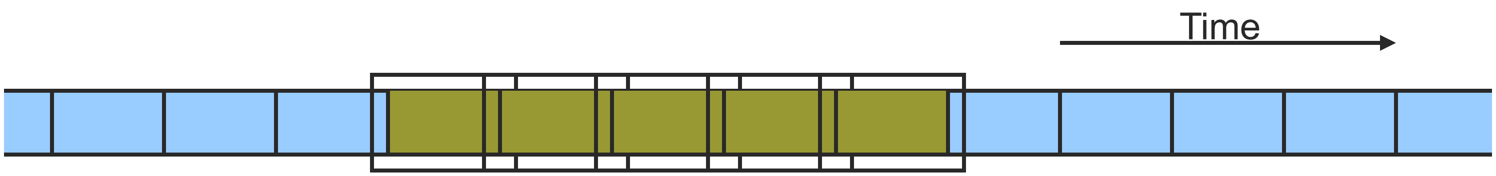
Goal



- Advantages:
  - More frames in pipe
  - Less time overall
  - Piggybacked ACKs

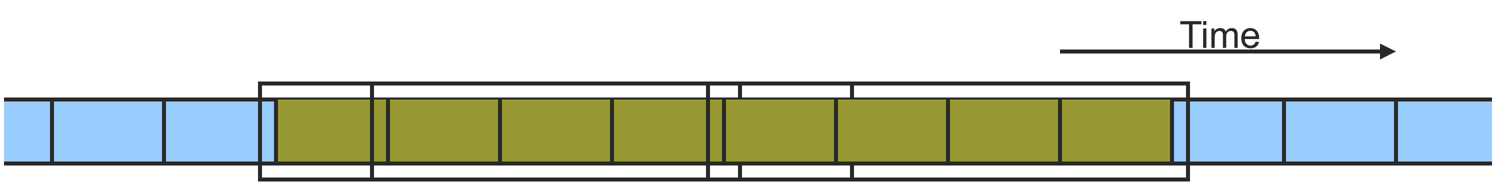
Concepts

- Consider an ordered stream of data frames
- Stop-and-Wait
  - Window of one frame
  - Slides along stream over time



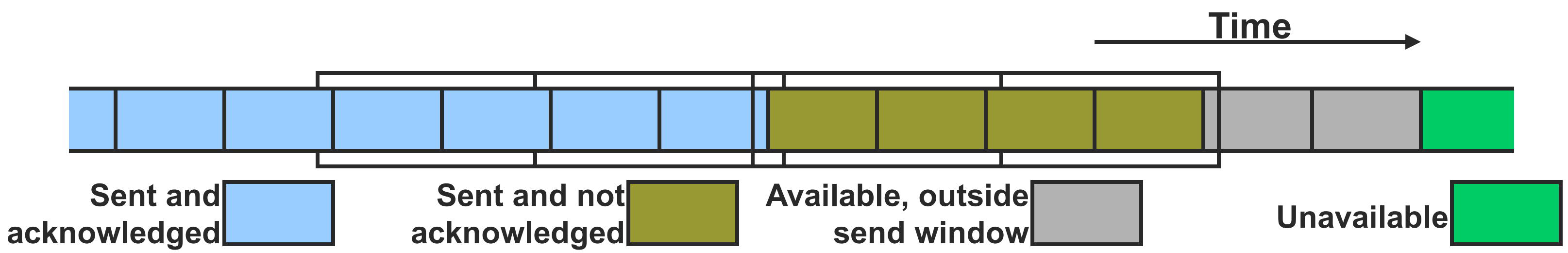
Concepts

- Sliding Window Protocol
  - Multiple-frame send window
  - Multiple frame receive window



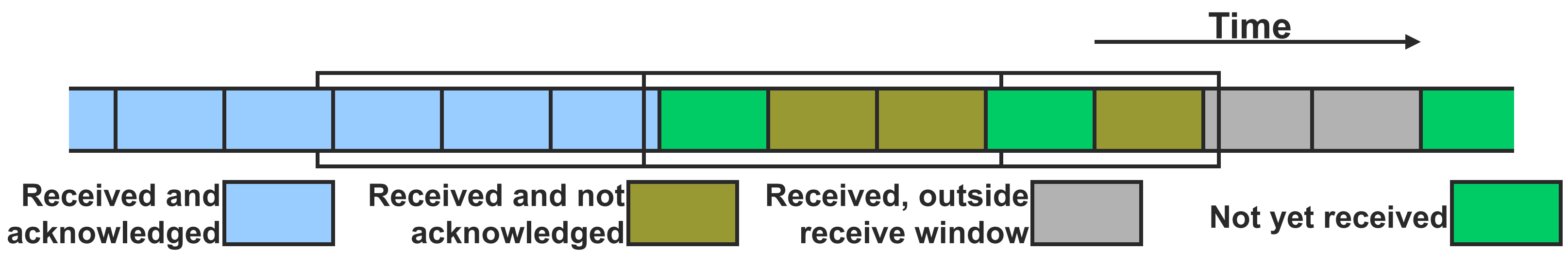
Sliding Window

- Send Window
  - Fixed length
  - Starts at earliest unacknowledged frame
  - Only frames in window are active



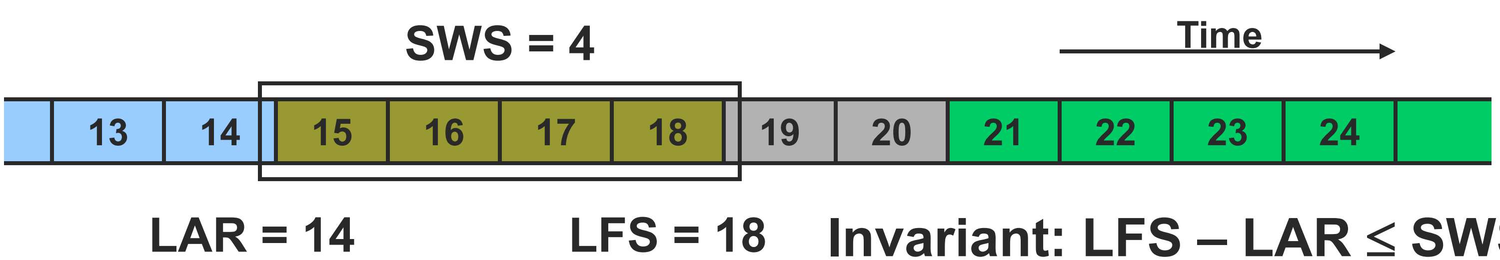
Sliding Window

- Receive Window
  - Fixed length (unrelated to send window)
  - Starts at earliest frame not received
  - Only frames in window accepted



Sliding Window Terminology

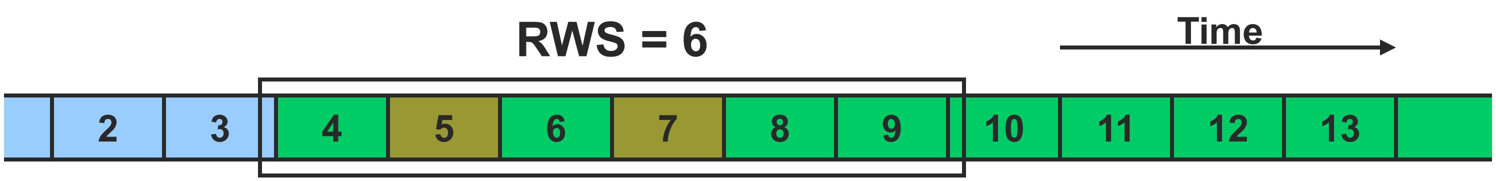
- Sender Parameters
  - Send Window Size (SWS)
  - Last Acknowledgement Received (LAR)
  - Last Frame Sent (LFS)



Invariant: LFS – LAR ≤ SWS

Sliding Window Terminology

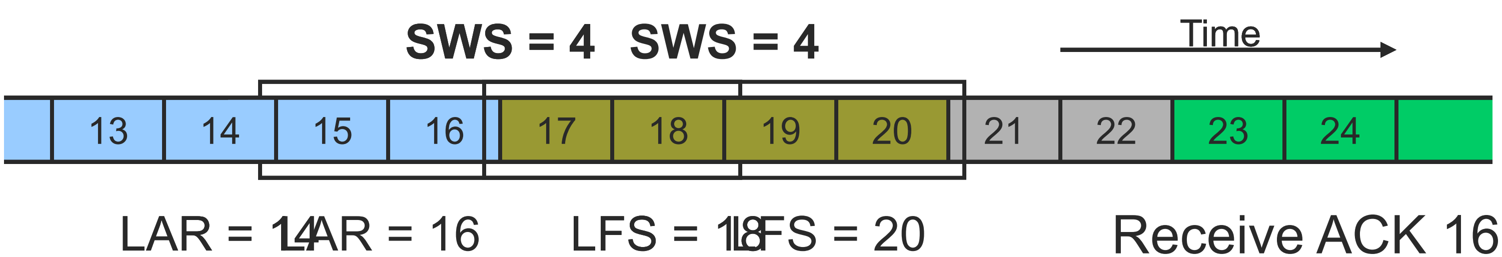
- Receiver Parameters
  - Receive Window Size (RWS)
  - Next Frame Expected (NFE)
  - Last Frame Acceptable (LFA)



NFE = 4 LFA = 9 **Invariant: LFA – NFE + 1 ≤ RWS**

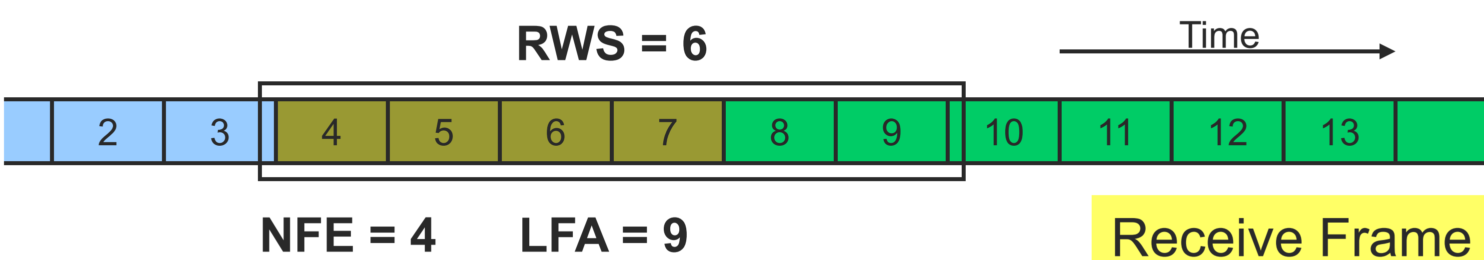
Sliding Window Details

- Sender Tasks
  - Assign sequence numbers
  - On ACK Arrival
    - Advance LAR
    - Slide window



Sliding Window Details

- Receiver Tasks
  - On Frame Arrival (N)
    - Silently discard if outside of window
      - N < NFE (NACK possible, too)
      - N >= NFE + RWS
    - Send cumulative ACK if within window

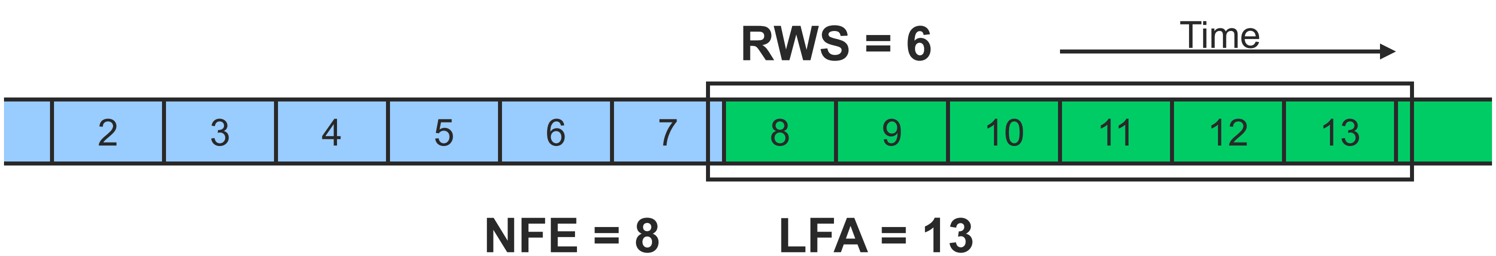


Receive Frame 4

Send ACK 7

Sliding Window Details

- Receiver Tasks
  - On Frame Arrival (N)
    - Silently discard if outside of window
      - N < NFE (NACK possible, too)
      - N >= NFE + RWS
    - Send cumulative ACK if within window



Sliding Window Details

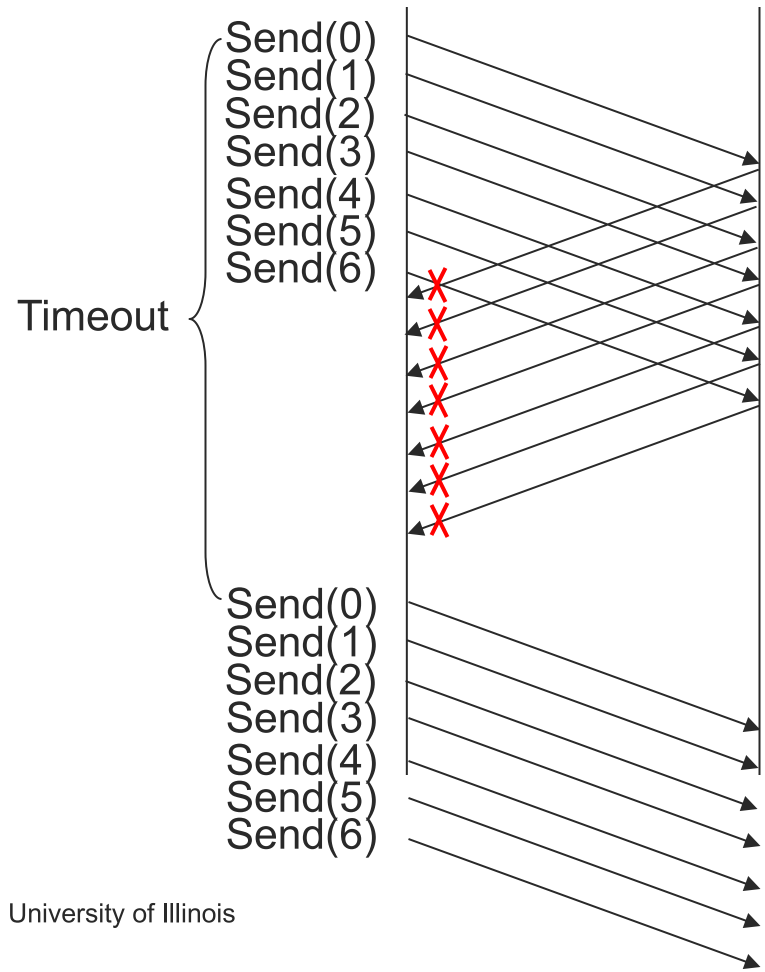
- Sequence number space
  - Finite number, so wrap around
  - Need space larger than SWS (outstanding frames)
    - In fact, need twice as large
- Example
  - 3-bit sequence numbers (0-7)
  - RWS = SWS = 7

Sliding Window Details

- Is $\log_2(SWS+1)$ bits enough?
  - No. Example:
  - 3-bit sequence numbers (0-7)
  - RWS = SWS = 7
  - Why isn't 3 bits enough (can you think of an example where it doesn't work?)

Sliding Window Details

- Example of incorrect behavior
  - 3-bit sequence numbers 0-7
  - RWS = SWS = 7
  - Sender transmits 0-6
  - All arrive, but ACK's lost
  - Sender retransmits
  - Receiver accepts as second incarnation of 0-6

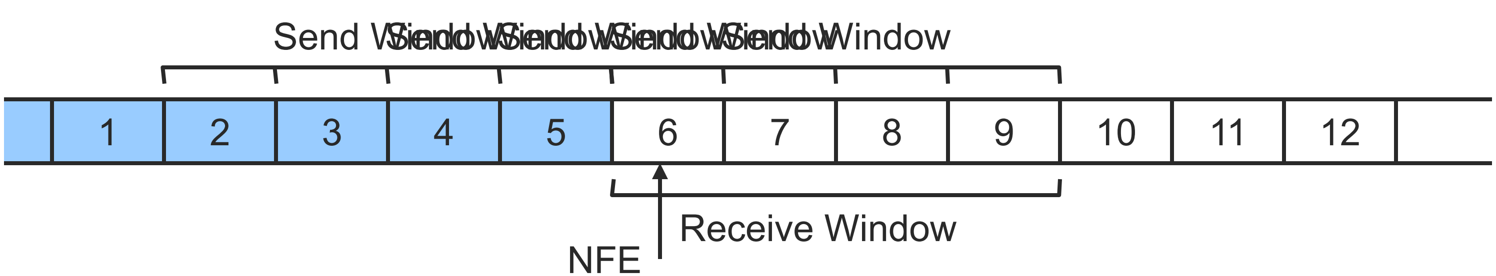


Sliding Window Sequence Numbers

- How many sequence numbers are necessary?
  - Key questions
    - Where can the send window be?
    - What frame can be received next?

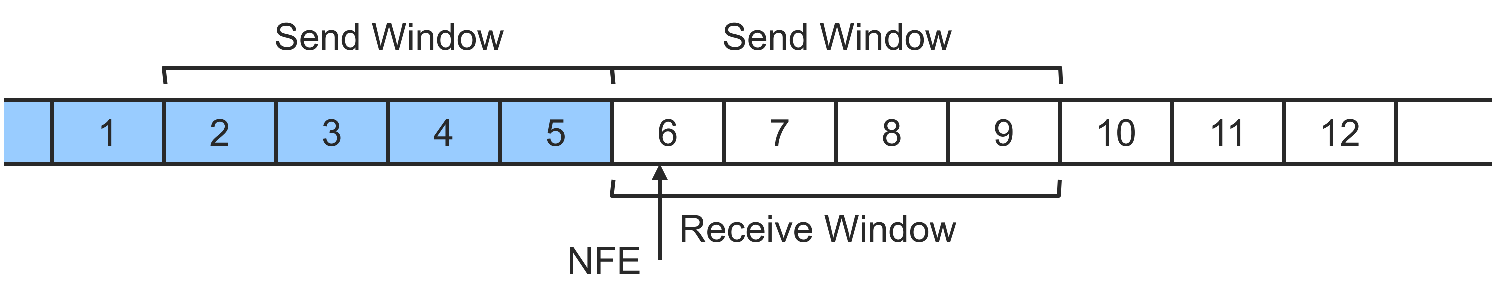
Sliding Window Sequence Numbers

- Assume SWS = RWS (simplest, and typical)
- Sender transmits full SWS
- Two extreme cases:
  - None received (waiting for 0...SWS - 1)
  - All received (waiting for SWS...2 SWS - 1)
- All possible packets must have unique sequence numbers



Sliding Window Sequence Numbers

- Extreme Locations for SWS
- Requirements
  - If a received packet is not in the receive window with no wrap, then it must not be in the receive window with wrap!
- Correctness condition:
  - Number of Sequence Numbers $\geq$ SWS + RWS
  - Alternates between two halves of the sequence number space



Sliding Window Sequence Numbers

- Example
  - If SWS = RWS = 8
  - At least 16 sequence numbers are needed
  - A 4-bit sequence number space is enough

- Warning
  - P&D sometimes uses the variable Max_Seq_Num for the number of sequence numbers and sometimes for the maximum sequence number (these differ by one!)
  - Use Num_Seq_Num for the number of sequence numbers: 0, 1, …, Num_Seq_Num – 1

Window Sizes

- How big should we make SWS?
  - Compute from delay x bandwidth
- How big should we make RWS?
  - Depends on buffer capacity of receiver

Delay x Bandwidth Product - Revisited

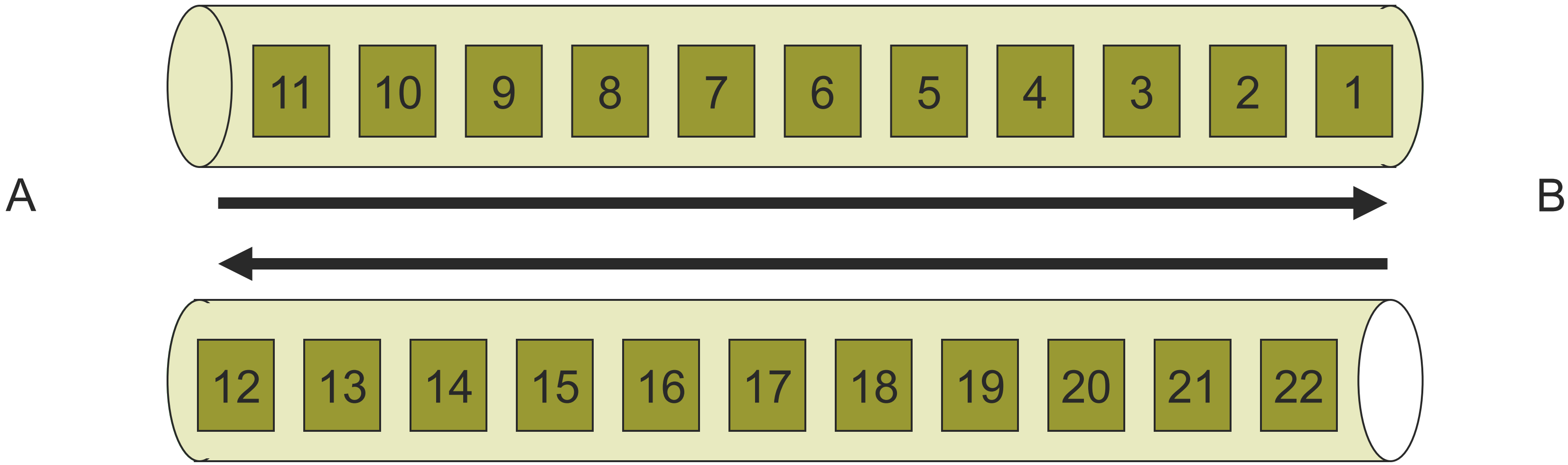
- Amount of data in “pipe”
  - channel = pipe
  - delay = length
  - bandwidth = area of a cross section
  - bandwidth x delay product = volume

Delay x Bandwidth Product

- Pipe
  - Half of data that must be buffered before sender responds to slowdown request

Delay x Bandwidth Product

- Bandwidth x delay product
  - How many bits the sender must transmit before the first bit arrives at the receiver if the sender keeps the pipe full
  - Takes another one-way latency to receive a response from the receiver



Sliding Window Protocol Code Example

- Parameters
  - last acknowledgement received (LAR)
  - last frame sent (LFS)
  - next frame expected (NFE)
  - last frame acceptable (LFA)

Sliding Window Protocol Code Example

- Constants
  - Rend/receive window size (SWS/RWS)
  - Maximum sequence number (MAX_SEQ_NO)
  - Frame size (FRAME_SIZE, constant for simplicity)

Sliding Window Protocol Code Example

- Data structures
  - Next frame expected (an integer)
  - One frame buffer for each entry in receive window
  - One presence bit for each entry
- Receive window cycles through
  - Sequence numbers
  - Data structures (thus RWS must divide MAX_SEQ_NO)

Sliding Window Protocol Code Example

```
#define RWS              8       /* receive window size    */
#define MAX_SEQ_NO       16      /* max. sequence number+1 */
                                 /* (must be multiple of   */
                                 /*  RWS for this code)    */
#define FRAME_SIZE       1000    /* constant for simplicity*/

char buf[RWS][FRAME_SIZE];       /* RWS frame buffers        */
int present[RWS];                /* are frame buffers full?*/
                                 /*   (initialized to 0's) */
int NFE = 0;                     /* next frame expected    */
extern void send_ack (int seq_no);
extern void pass_to_app (char* data);
void recv_frame (char* data, int seq_no);
```

Sliding Window Protocol Code Example

```
void recv_frame (char* data, int seq_no)
{
    int idx;            /* index into data structures */
    int i;              /* loop index                 */

    /* Map sequence numbers NFE...predecessor (NFE)
       into 0...MAX_SEQ_NO - 1, then see if seq_no
       falls within the receive window. */
    if (seq_no - NFE)< RWS)
                  /* Frames outside the window */
                  /* are ignored. (but an ACK  */
                  /* is sent; why?)            */
```

Sliding Window Protocol Code Example

```
/* Calculate index into data structures. */
idx = (seq_no % RWS);

if (!present[idx]) {/* frame is not dup */
    present[idx] = 1;/* mark received        */
    memcpy (buf[idx], data, FRAME_SIZE);
                     /* copy data into buf  */
```

Sliding Window Protocol Code Example

```
/* Got a new frame; pass frames up to host? */
for (i = 0; i < RWS; i++) {
    idx = (i + NFE) % RWS;      /* Re-use idx.*/
    /* first missing frame becomes NFE */
    /* after this loop terminates */
    if (!present[idx]) break;

    /* Frame is present—send it up! */
    pass_to_app (buf[idx]);
    present[idx] = 0; /* Mark buffer empty. */
}
/* Advance NFE to first missing frame. */
NFE = NFE + i;
}
```

Sliding Window Protocol Code Example

```
        /* Frame handled (might have */
        /* been duplicate). */
    } /* (Send ACK for any frame received */

    /* Now send acknowledgement for              */
    /* predecessor (NFE).                        */
    send_ack (NFE - 1);
}
```

Correctness

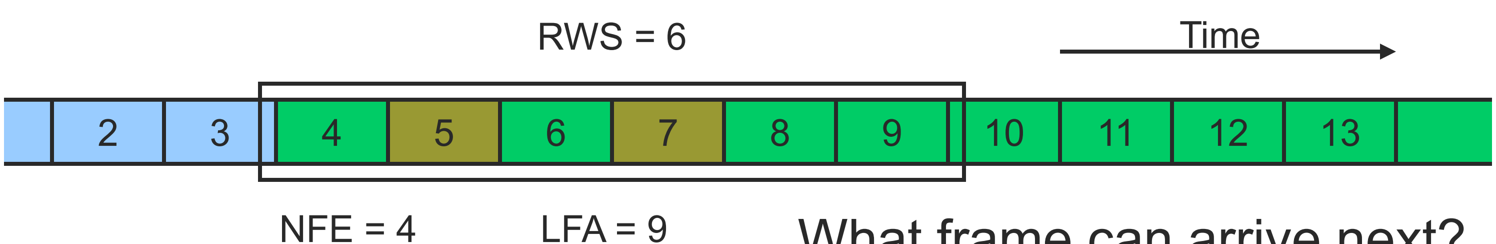
- Claim
  - A sliding window protocol leads to in-order delivery of all frames
- Assumptions
  - All sequence numbers are different
  - Frames can be lost
  - Frames can be delayed an arbitrarily finite amount of time
  - Frames are not reordered
  - Frames can arrive with detectable errors
- Are these assumption adequate?

Sliding Window Protocol Correctness

- Need one more assumption
  - Any given frame is received without errors after a finite number of retransmissions
- Proof in two steps
  - Establish correctness assuming infinite sequence number space
  - Show that finite sequence number space does not affect result as long as it has >= 2 max (SWS, RWS) possible numbers

Sliding Window Protocol Correctness

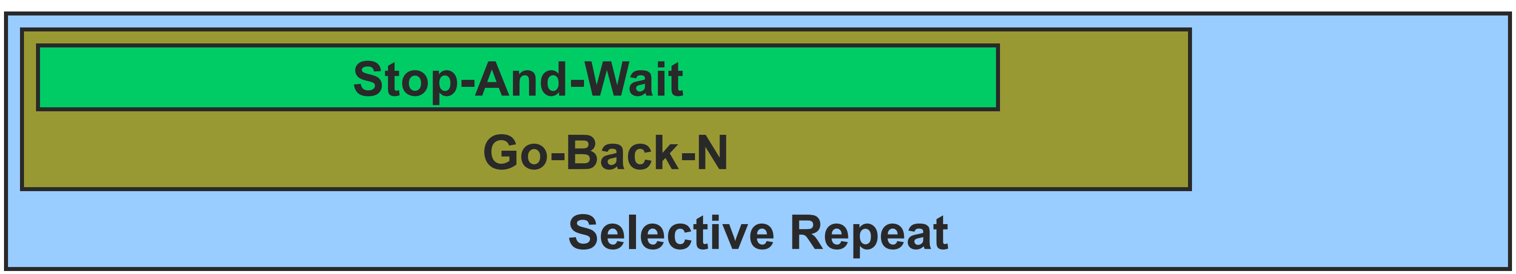
- Step 1: establish correctness assuming infinite sequence number space
  - Use induction on k with invariant "the k^{th} frame is eventually received"
- Step 2: show that finite sequence number space does not affect result as long as it has >= 2 max (SWS, RWS) possible numbers



What frame can arrive next?

ARQ Algorithm Classification

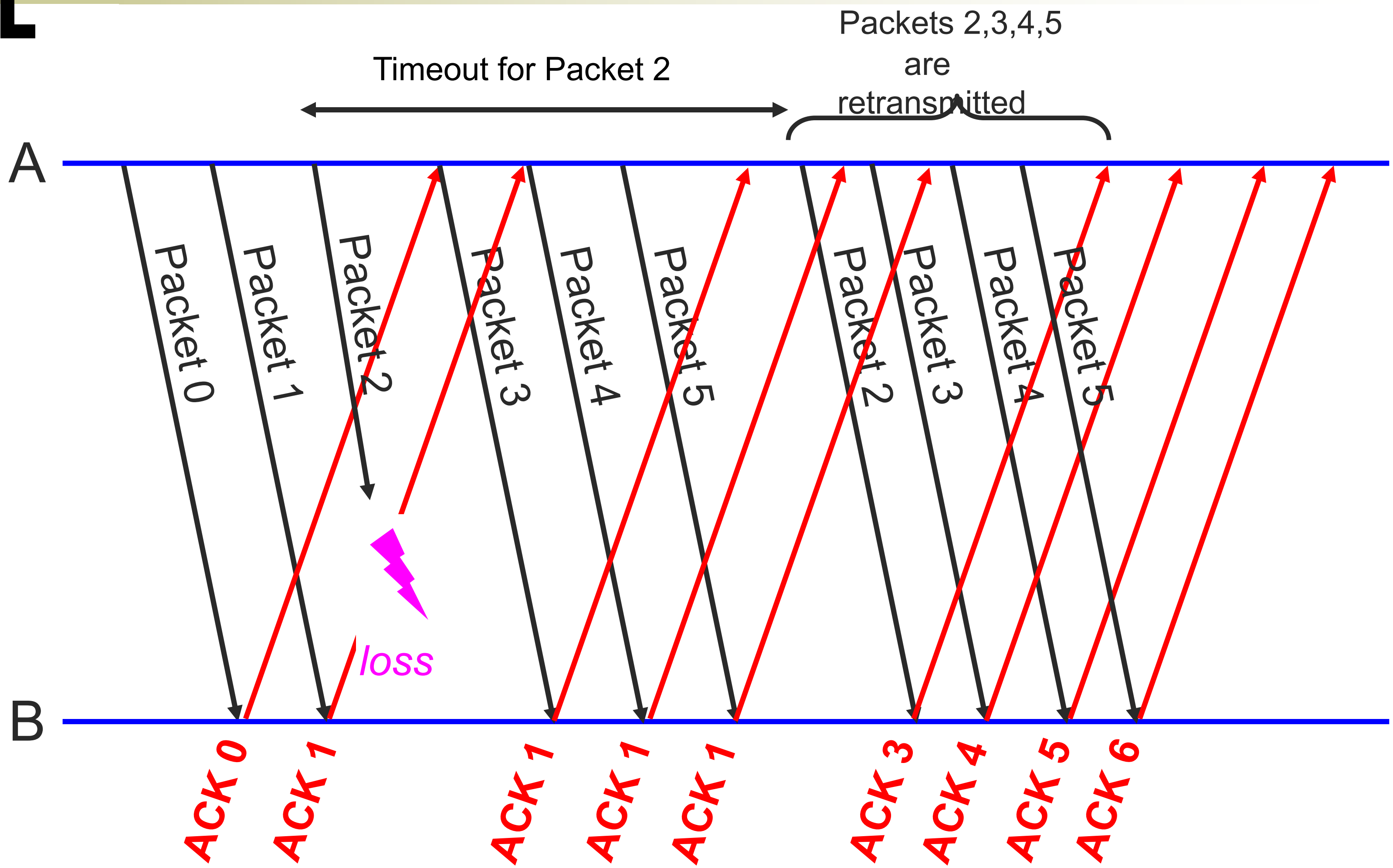
- Three Types:
  - Stop-and-Wait: SWS = 1 RWS = 1
  - Go-Back-N: SWS = N RWS = 1
  - Selective Repeat: SWS = N RWS = M
    - Usually M = N



Sliding Window Variations: Go-Back-N

- SWS = N, RWS = 1
- Receiver only buffers one frame
- If a frame is lost, the sender may need to retransmit up to N frames
  - i.e., sender “goes back” N frames
- Variations
  - How long is the frame timeout?
  - Does receiver send NACK for out-of-sequence frame?

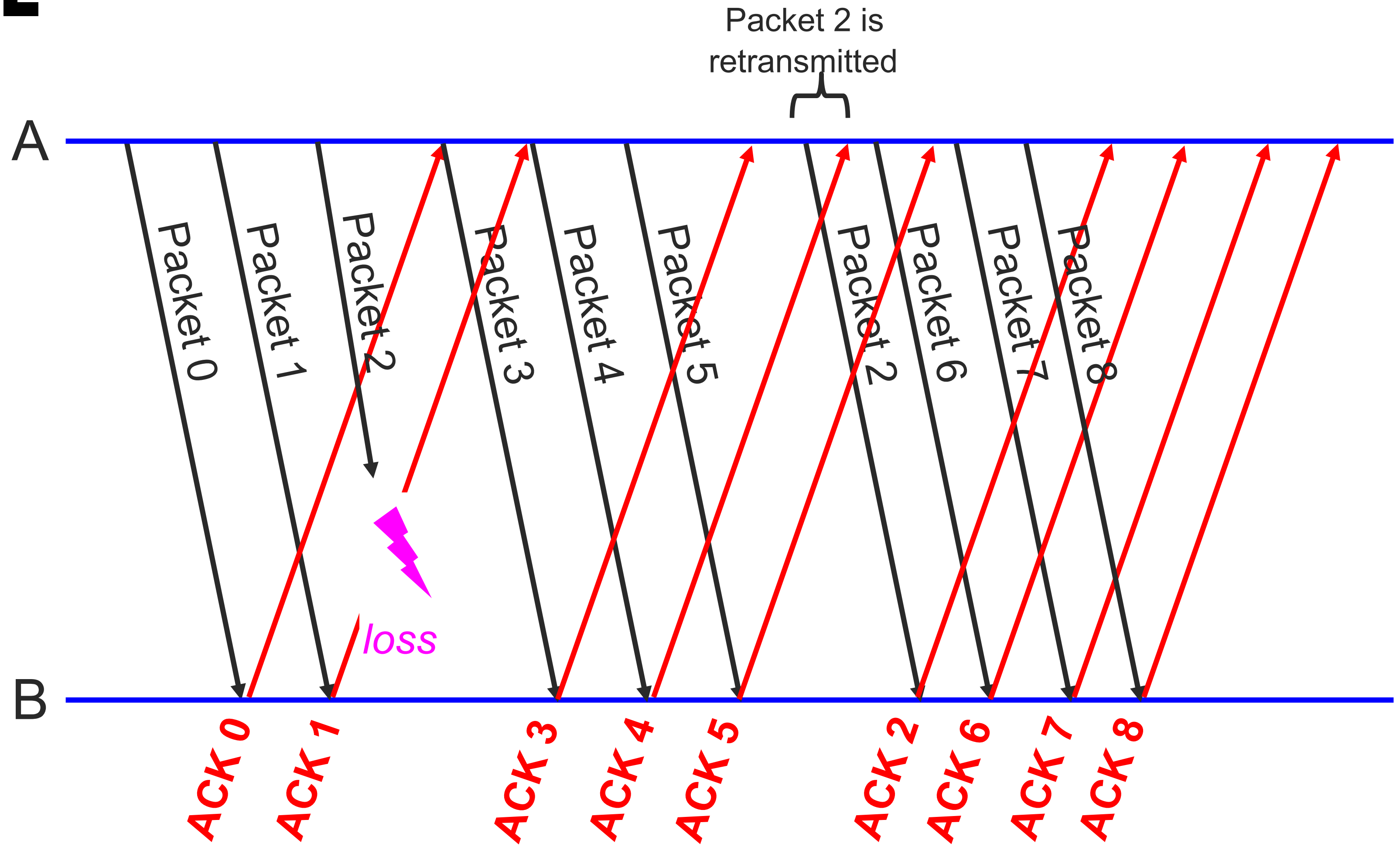
Go-Back-N: Cumulative ACKs



Sliding Window Variations: Selective Repeat

- SWS = N, RWS = M
- Receiver buffer M frames
- If a frame is lost, sender must only resend
  - Frames lost within the receive window
- Variations
  - How long is the frame timeout?
  - Use cumulative or per-frame ACK?
  - Does protocol adapt timeouts?
  - Does protocol adapt SWS and/or RWS?

Selective Repeat



Roles of a Sliding Window Protocol

- Reliable delivery on an unreliable link
  - Core function
- Preserve delivery order
  - Controlled by the receiver
- Flow control
  - Allow receiver to throttle sender

- Separation of Concerns
  - Must be able to distinguish between different functions that are sometimes rolled into one mechanism

Forward Error Correction (FEC)

- Alternative to ARQ algorithms
- Idea
  - Error correction instead of error detection
  - Send extra information to avoid retransmission (i.e., fix errors first/forward rather than afterward/backward)
- Why
  - Very high latency connections
  - Difficult for retransmission