Pre-Class Discussion

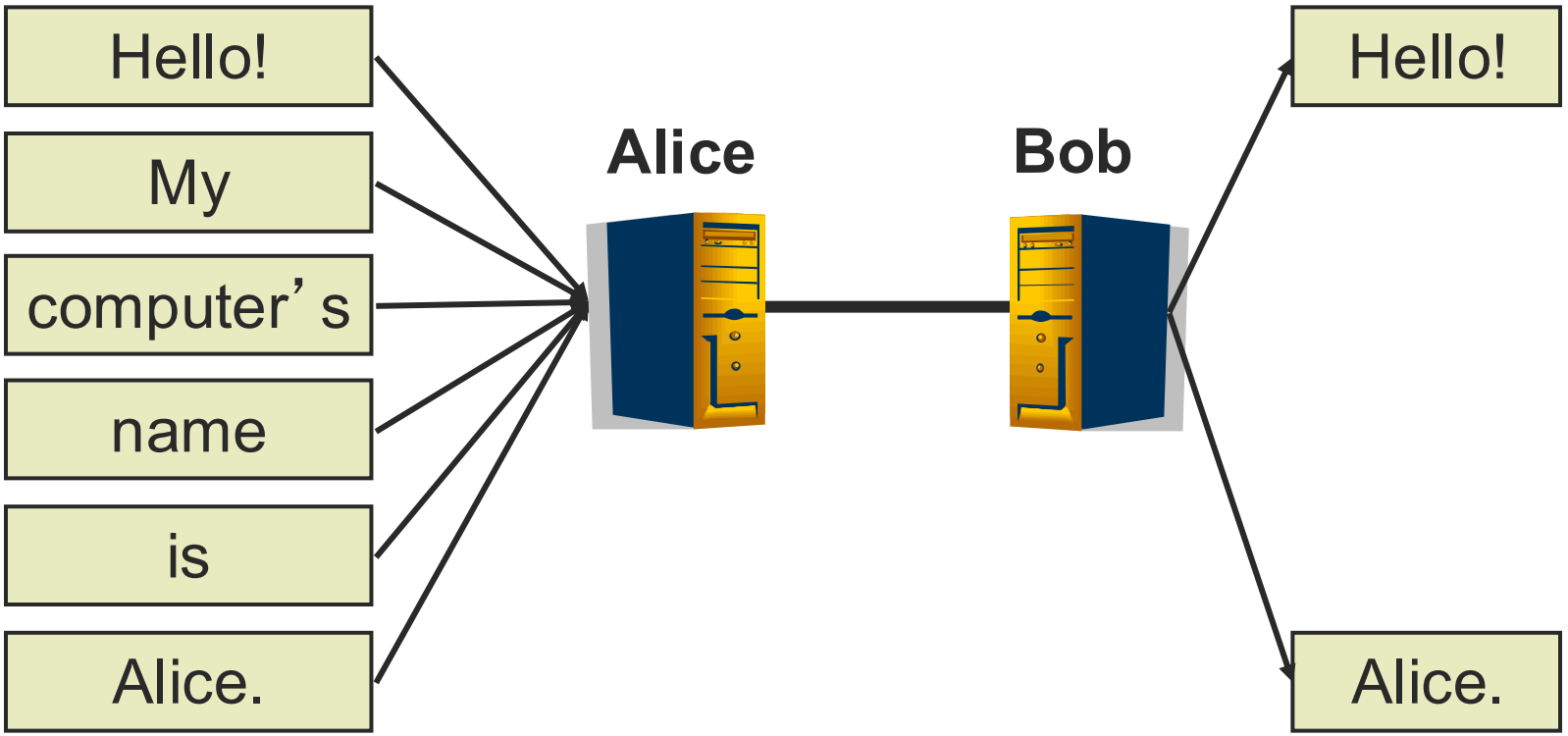
- Talk to the person next to you
- Discuss how humans achieve “reliable” communication over the phone
- Prepare to share your team’s ideas

Reliable Transmission

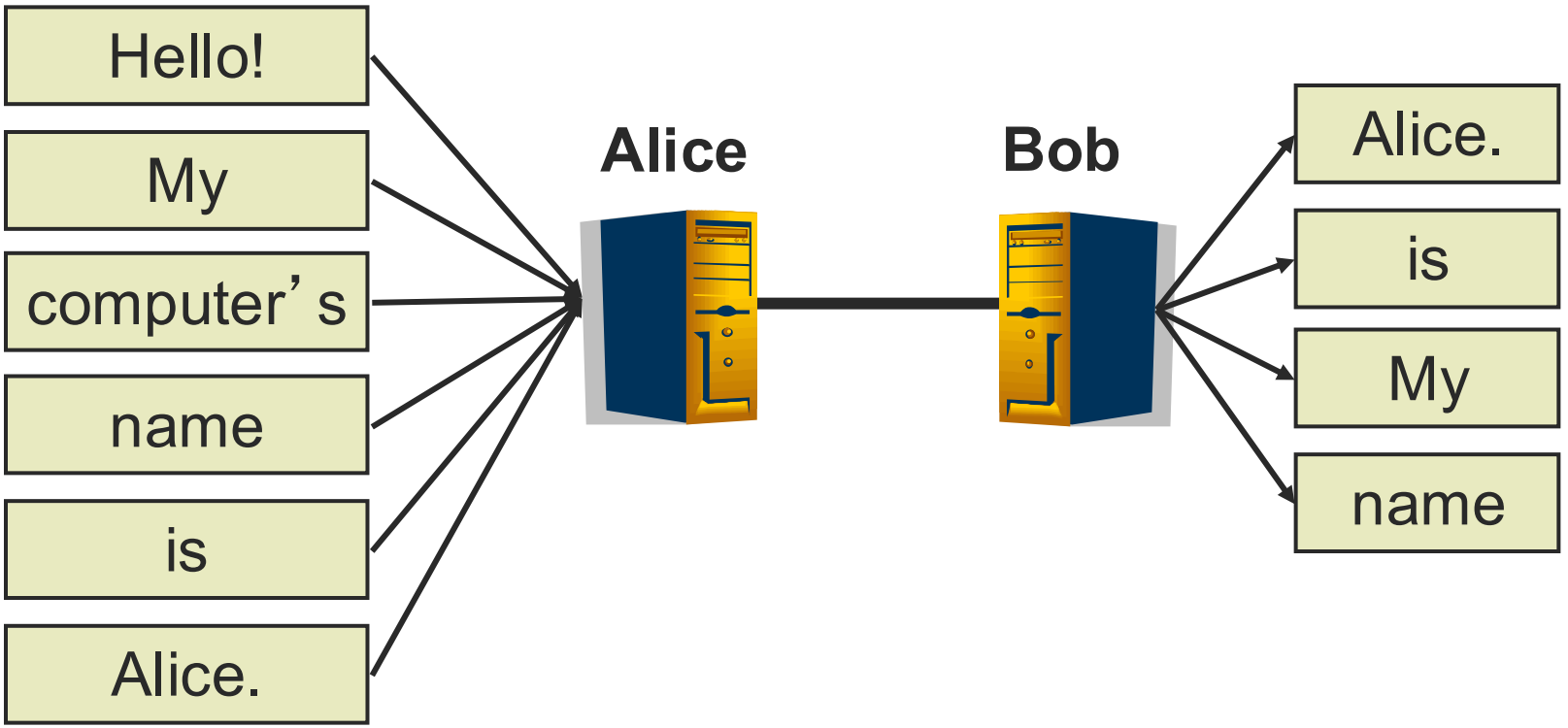
Learning Objectives

- Reliable Transmission
  - Stop-and-Wait
  - Sliding Window
    - Go-Back-N
    - Selective Repeat

Reliable Transmission



Reliable Transmission



Reliable Transmission

- Suppose error detection identifies valid and invalid packets
  - Corrupted packets are discarded
- Can we make an unreliable channel appear reliable?
  - Ensure packet delivery
  - Maintain packet order
  - Provide reliability at full link capacity

Reliable Transmission Outline

- Fundamentals of Automatic Repeat reQuest (ARQ) algorithms
  - Provide reliability based on acknowledgements and timeouts
  - Sender retransmits unacknowledged packets after timeouts
- ARQ algorithms (simple to complex)
  - stop-and-wait
  - concurrent logical channels
  - sliding window
    - go-back-n
    - selective repeat

Terminology

- Acknowledgement (ACK)
  - Receiver tells the sender when a packet is received
    - Cumulative acknowledgement (ACK)
      - Has received everything up to a specified packet
    - Selective acknowledgement (SACK)
      - Specifies set of packets received
    - Negative acknowledgement (NAK)
      - Specifies which packets were *not* received

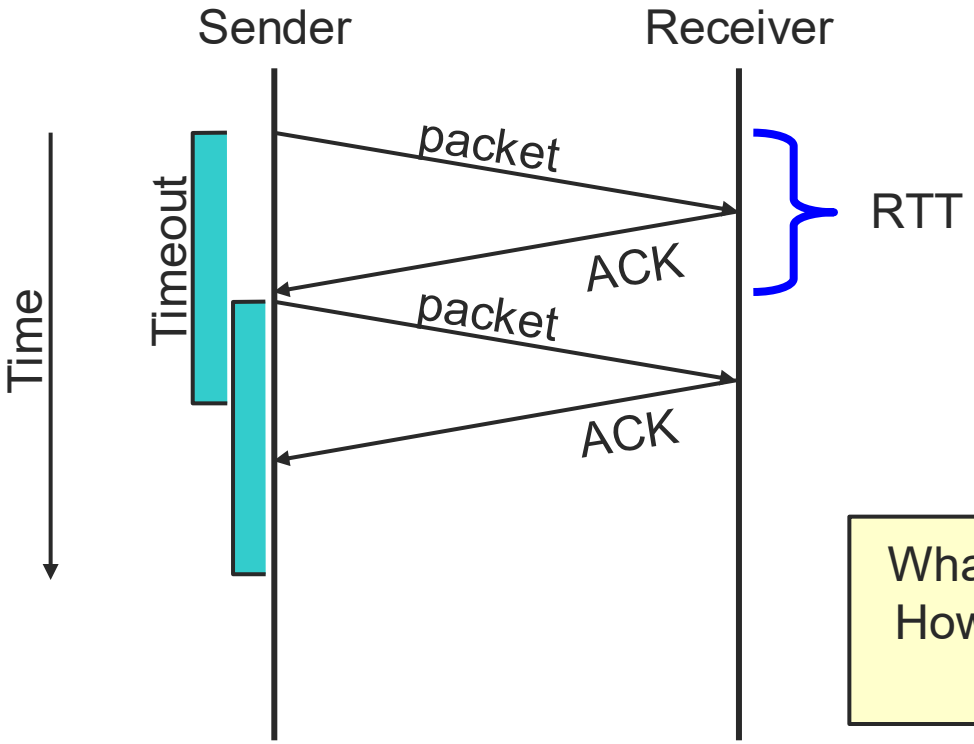
Terminology

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

Stop-and-Wait

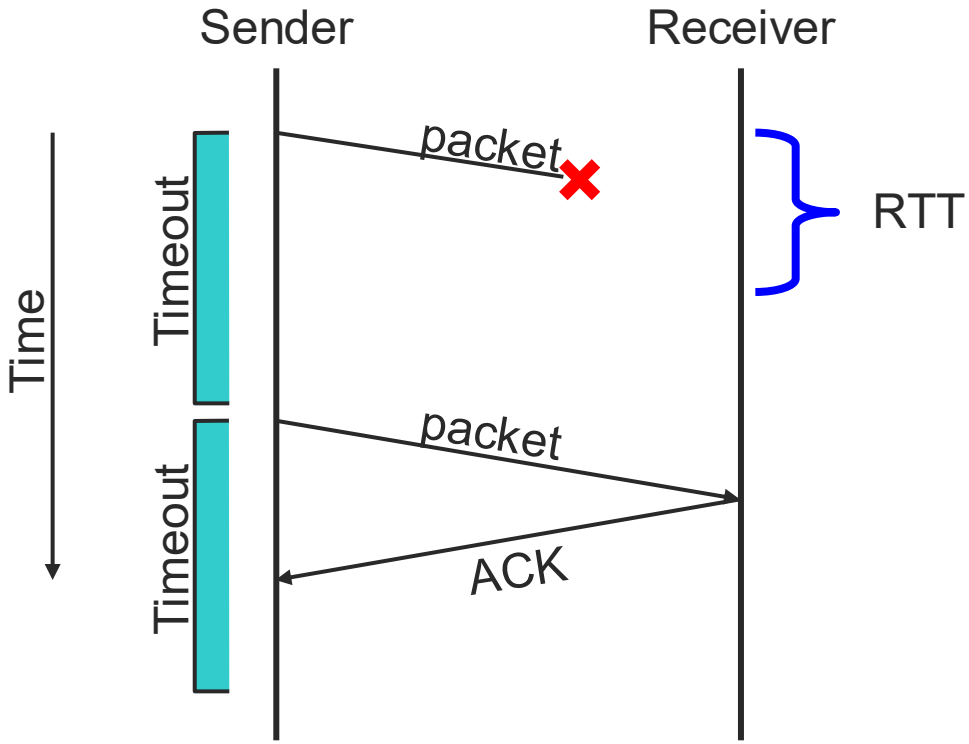
- Goal:
  - Guaranteed, at-most-once delivery
- Protocol Challenges
  - Dropped (or corrupted) packet/ACK
  - Duplicate packet/ACK
- How it works
  1. Send a packet
  2. Wait for an ACK or timeout
  3. If timeout, go to 1 (to retransmit)
  4. If ACK, get new packet, go to 1

Stop-and-Wait: Success



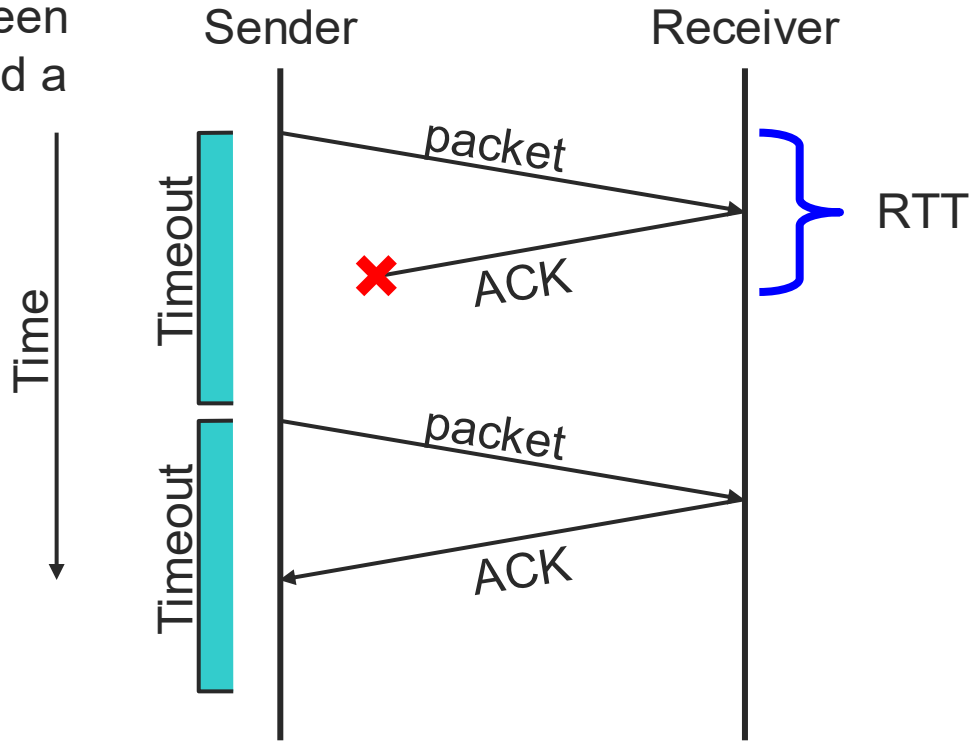
What can go wrong? How will it affect our protocol?

Stop-and-Wait: Lost packet



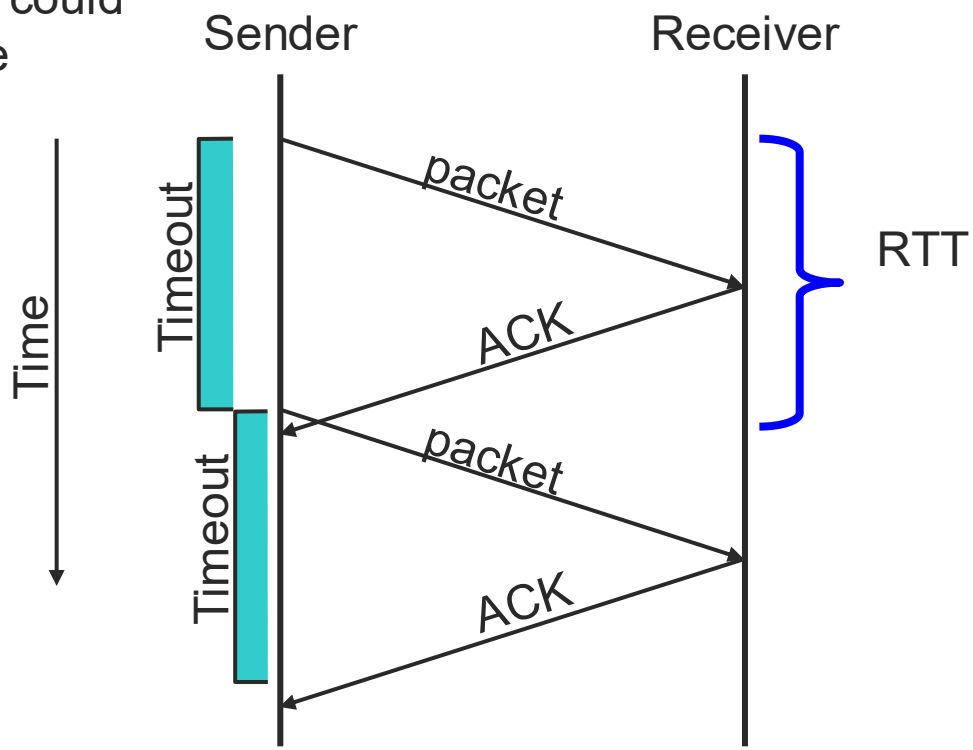
Stop-and-Wait: Lost ACK

Sender can't tell the difference between a lost packet and a lost ACK

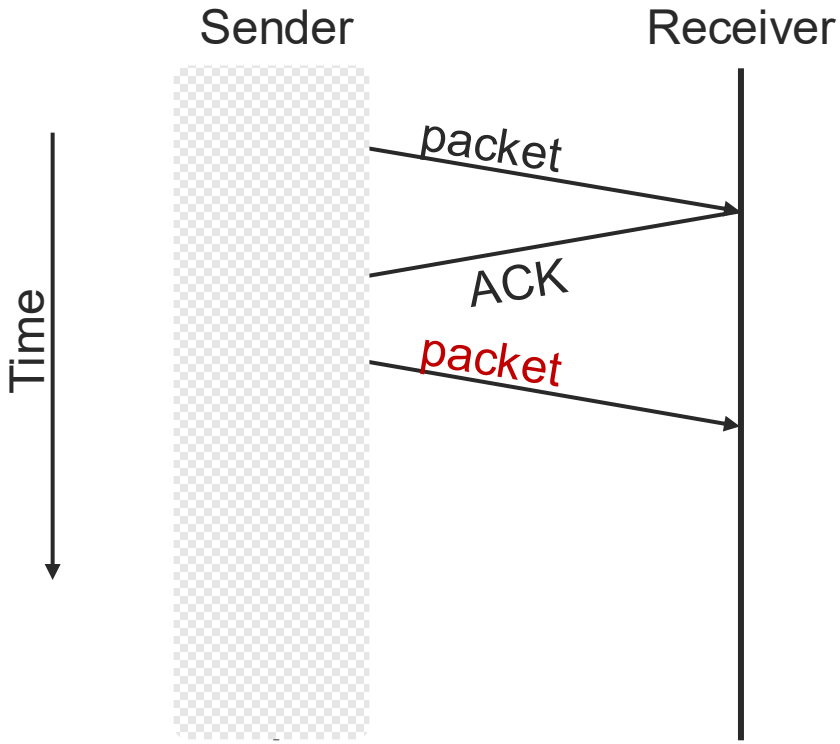


Stop-and-Wait: Delayed Packet

Delayed packets could trigger premature retransmissions



Stop-and-Wait: Receiver's View



Receiver doesn't know if ACK is lost or delivered.

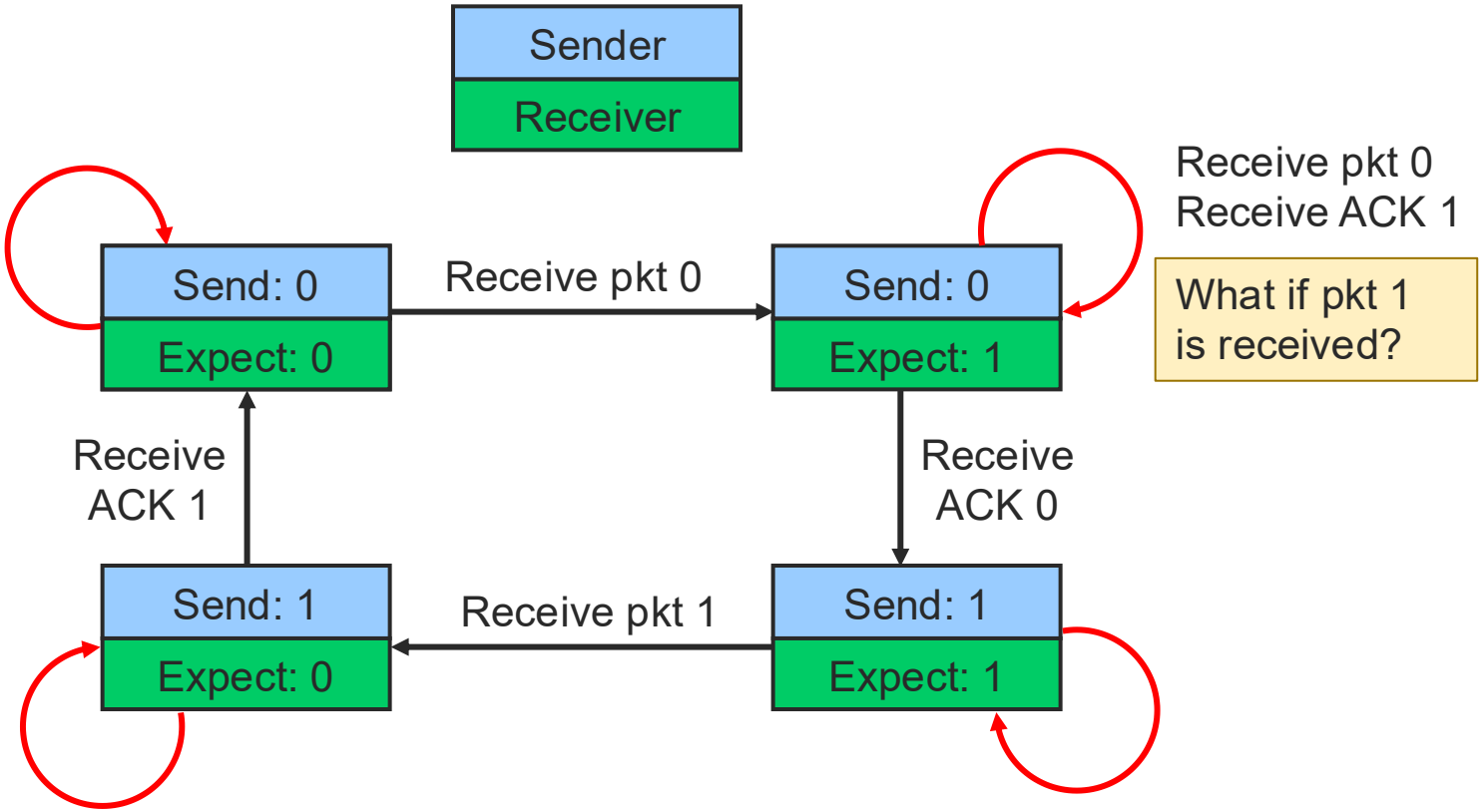
Is this a retransmitted and new packet?

Solution:
Sequence Numbers

Stop-and-Wait

- Packet sequence numbers
  - sender tracks packet ID to send
  - receiver tracks next packet ID expected
- If no reordering
  - How many bits are required for the sequence number?
  - 1 bit will suffice

Alternating-Bit Protocol: Simplified State Diagram



Stop-and-Wait

- Packet sequence numbers
  - sender tracks packet ID to send
  - receiver tracks next packet ID expected
- If no reordering
  - How many bits are required for the sequence number?
  - 1 bit will suffice
- What if the network reorders packets?
  - More bits are needed

Stop-and-Wait

- We have achieved
  - packets delivered reliably and in order
  - Is that enough?
- Problem
  - Only allows one outstanding packet
    - Does not keep the pipe full
  - Example
    - 100ms RTT
    - One packet per RTT = 1500B
    - 120 kbps only! (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 fall behind the others

ARQ: Where are We?

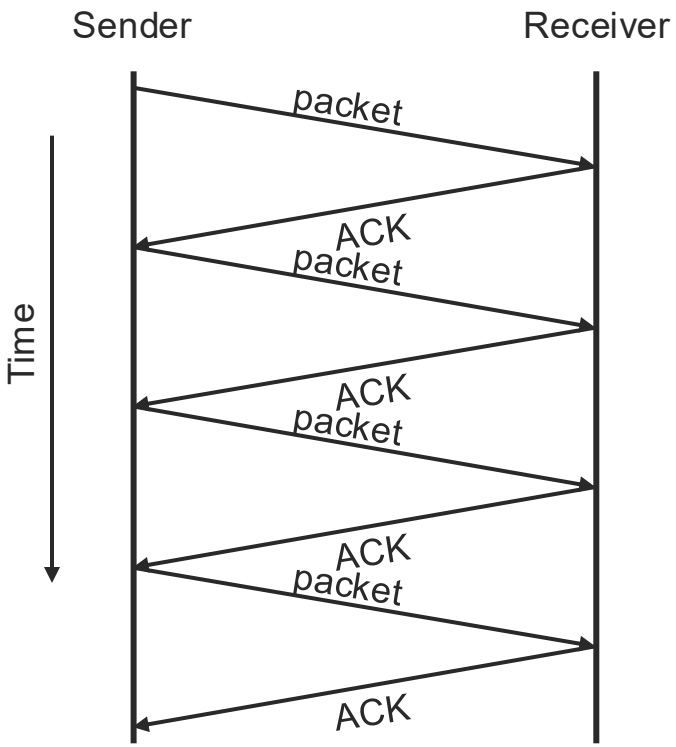
- Goals for reliable transmission
  - Make channel appear reliable
  - Maintain packet order
  - Impose low overhead/allow full use of link
- Stop-and-Wait
  - Provides reliable in-order delivery
  - Sacrifices performance
- Concurrent Logical Channels
  - Provides reliable delivery at full link bandwidth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

Sliding Window Protocol

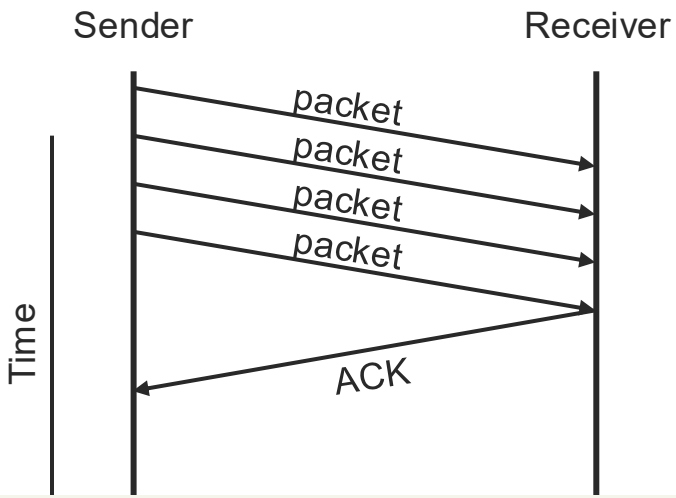
- General, pipelined ARQ algorithm
- Most important and used by TCP
- Outline
  - Concepts
  - Terminology
  - Details
  - Schemes
    - Go-Back-N, Selective Repeat
  - Code example
  - Proof of eventual in-order delivery

Keeping the Pipe Full

Stop-and-Wait



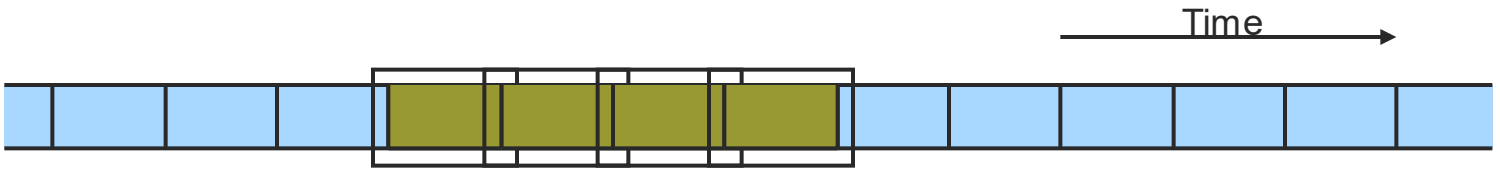
Goal



- Advantages:
  - More packets in pipe
  - Less time overall
  - Piggybacked, cumulative ACKs

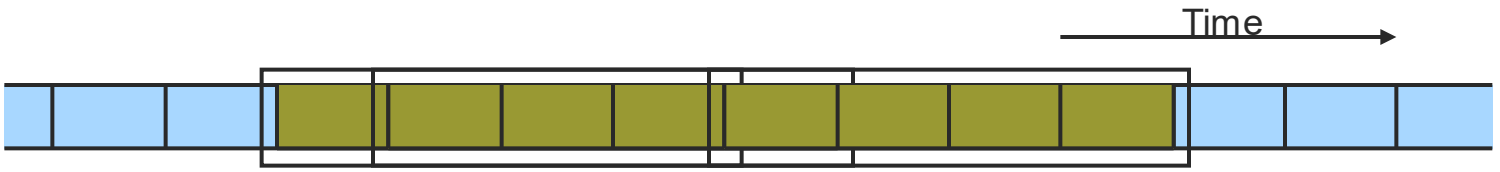
Concepts

- Consider an ordered stream of data packets
- Stop-and-Wait
  - Window of one packet to deliver
  - Slides along stream over time



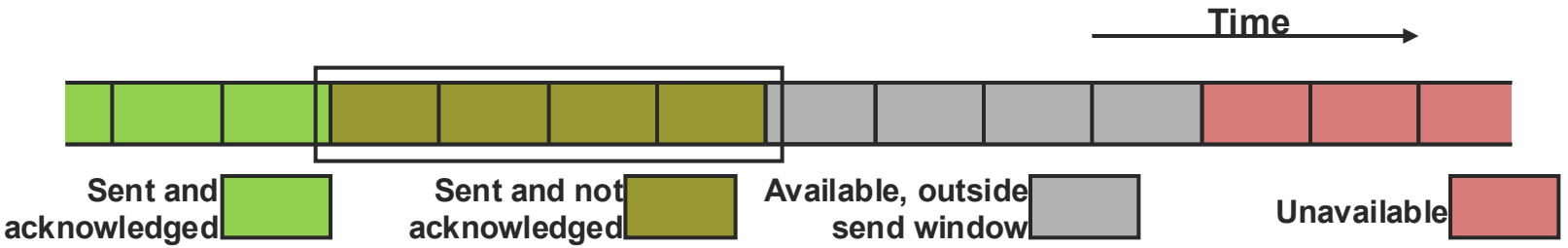
Concepts

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



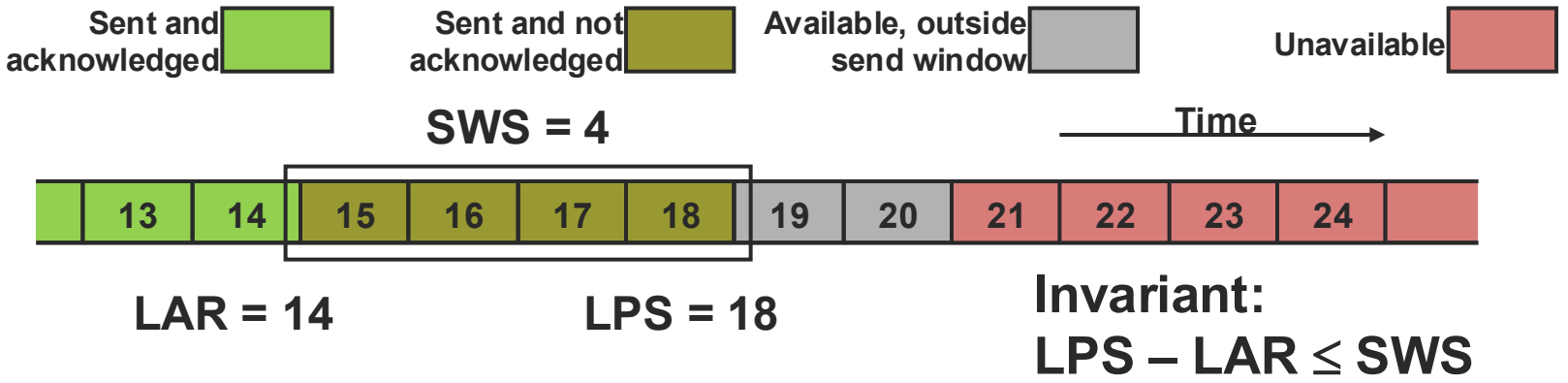
Sliding Window: Sender

- Send Window
  - Fixed length
  - Starts at earliest unacknowledged packet
  - Only packets in window are active



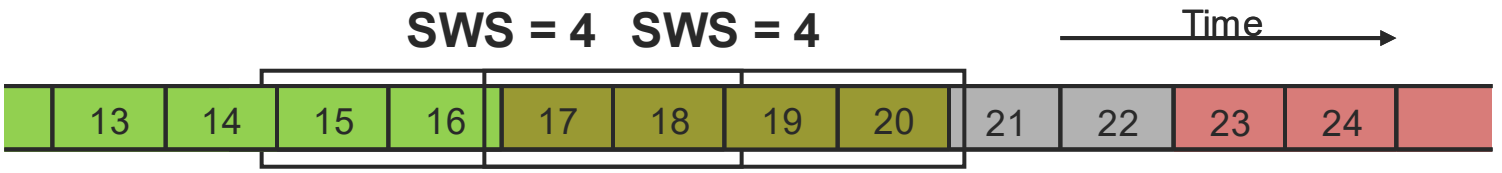
Sliding Window: Sender

- Sender Parameters
  - Send Window Size (SWS)
    - Maximum outstanding/unacknowledged packets
  - Last ACK Received (LAR)
  - Last Packet Sent (LPS)



Sliding Window: Sender

- Sender Tasks
  - Assign sequence numbers
  - On ACK arrival
    - Advance LAR and slide window
  - Maintain retransmission/timeout timers



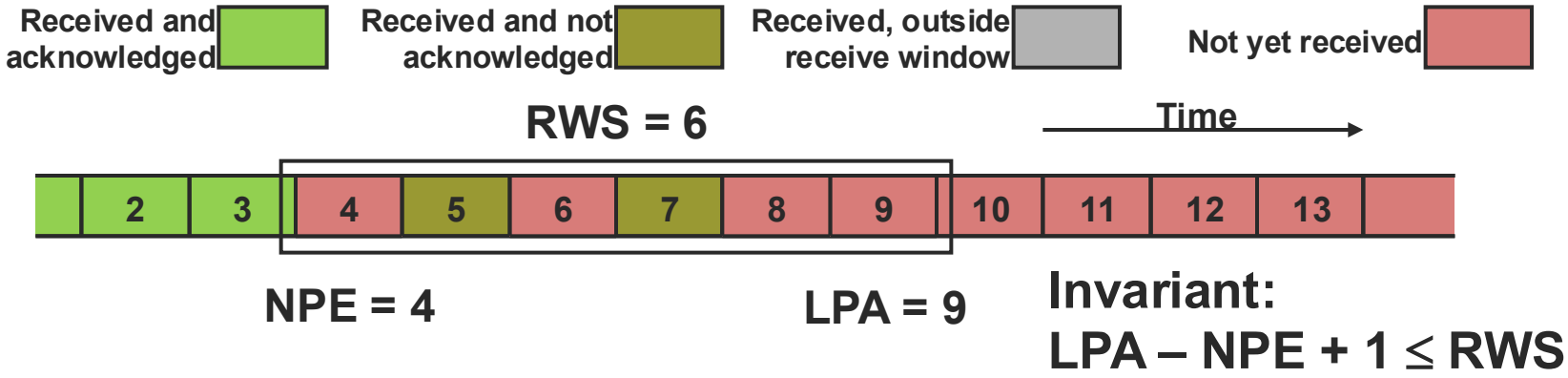
Receive (cumulative) ACK 16

Sliding Window: Receiver

- Receive Window (unrelated to send window)
  - Fixed length
  - Starts at earliest packet not received
  - Only packets in window are accepted

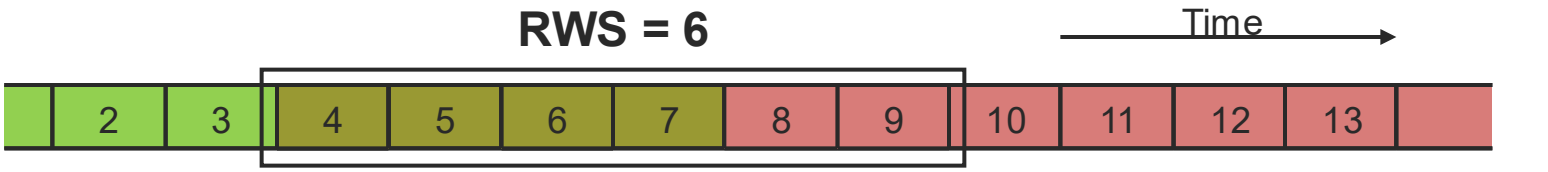
Sliding Window: Receiver

- Receiver Parameters
  - Receive Window Size (RWS)
    - Maximum out-of-order packets that receiver accepts
  - Next Packet Expected (NPE)
  - Last Packet Acceptable (LPA)



Sliding Window: Receiver

- Receiver Tasks
  - On packet Arrival (N)
    - Silently discard if outside of window
      - N < NPE or N > LPA
    - Send cumulative ACK if within window

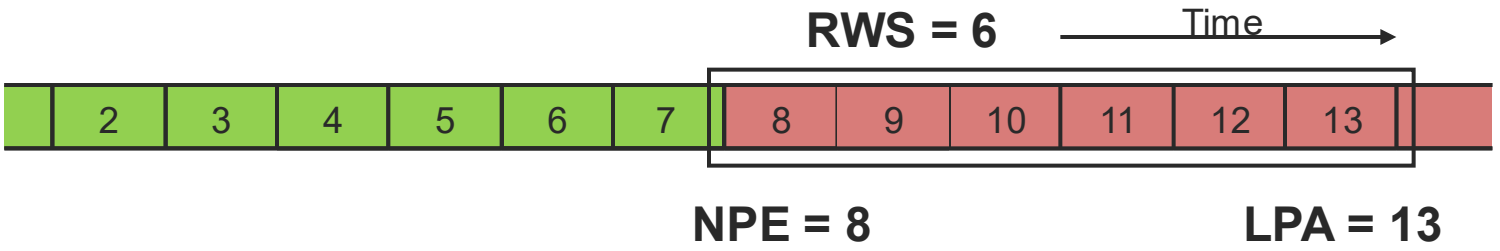


Receive packet 4

Send ACK 7

Sliding Window: Receiver

- Receiver Tasks
  - On packet Arrival (N)
    - Silently discard if outside of window
      - N < NPE or N > LPA
    - Send cumulative ACK if within window



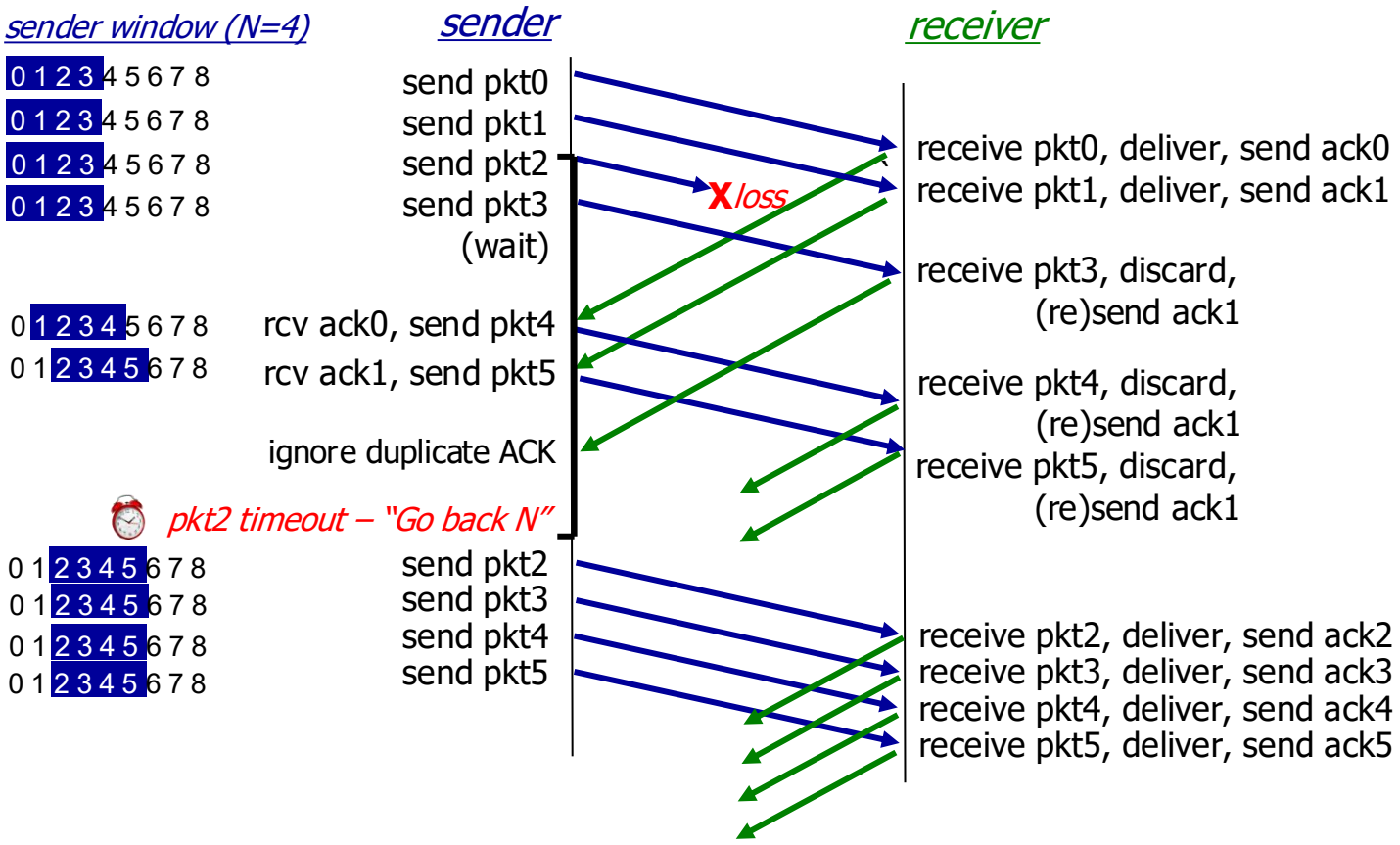
ARQ Algorithm Classification

- 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
    - Not helpful for M > N
  - Each type may have multiple variants
    - e.g., cumulative or selective ACKs, window sizes

Sliding Window Variations: Go-Back-N

- SWS = N, RWS = 1
- Receiver only buffers one packet and sends a cumulative ACK
  - discards out-of-order packets
- If a packet is lost, the sender may need to retransmit up to N packets
  - i.e., sender “goes back” N packets

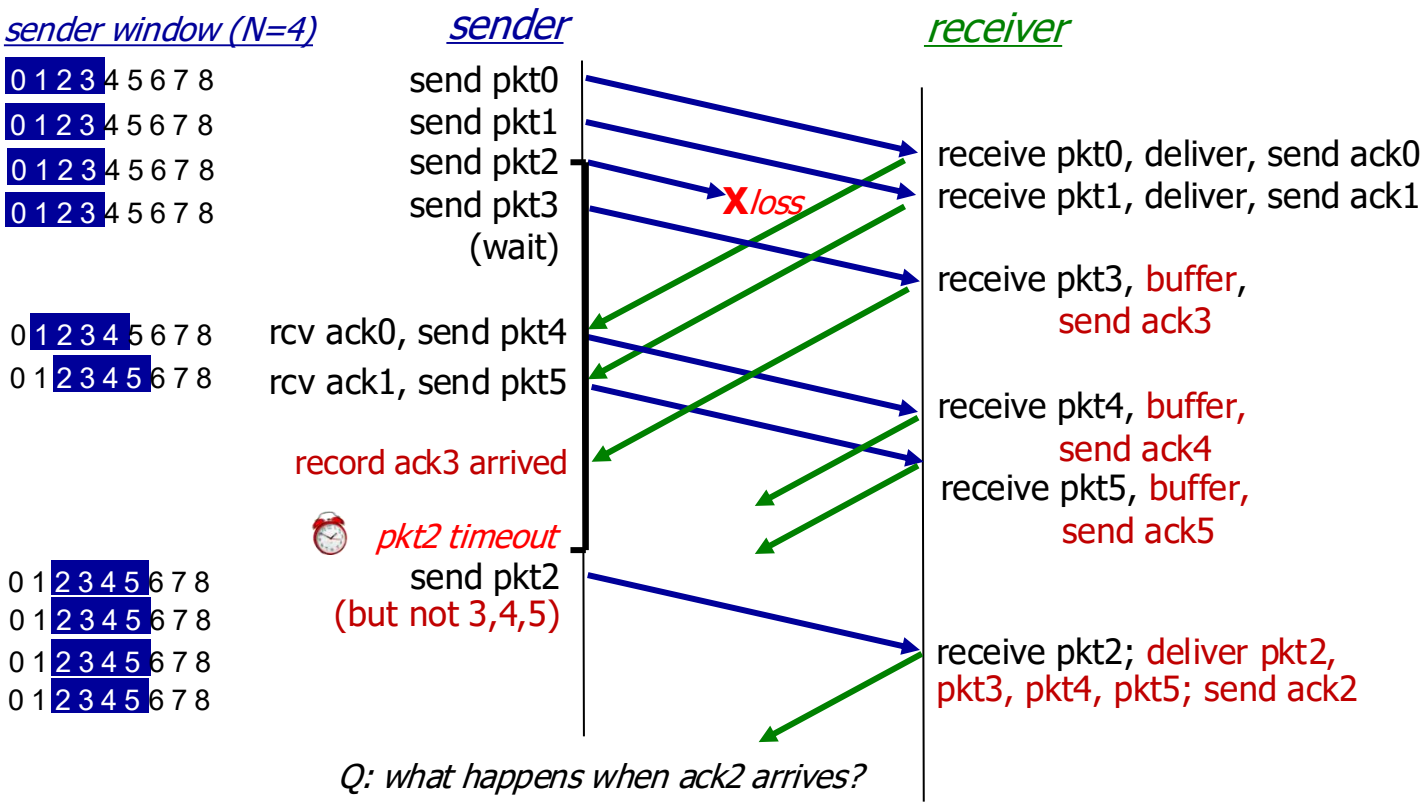
Go-Back-N (vanilla version) Example



Sliding Window Variations: Selective Repeat

- SWS = N, RWS = M
- Receiver buffers M packets and individually acknowledges received packets
- Sender times out and retransmits individually for unacknowledged packets
  - sender maintains timer for each unacknowledged packet

Selective Repeat (vanilla version) Example



Roles of a Sliding Window Protocol

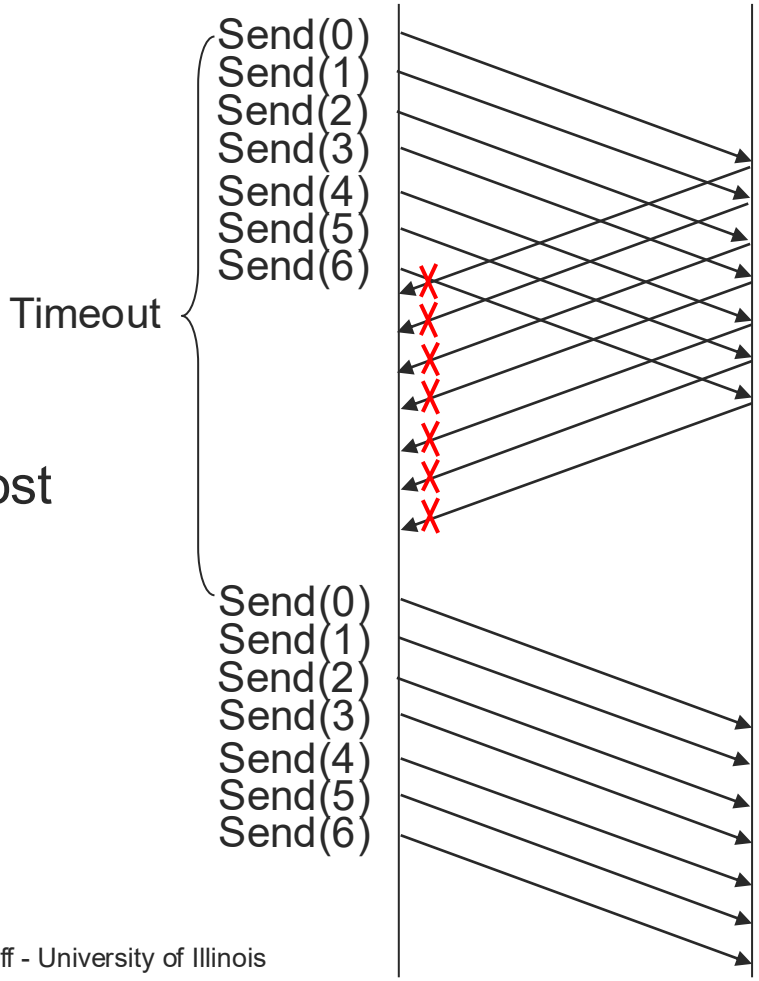
- Reliable delivery on an unreliable link
  - Core function
- Preserve delivery order
  - Controlled by the receiver
- Flow control
  - Allow receiver to throttle sender

Sequence Number Space

- Sequence number space
  - Finite number, so wrap around
- Need space at least larger than SWS
  - to distinguish retransmitted/new packets
  - e.g., 1 bit space for stop-and-wait
- TotalSeqNum ≥ SWS + 1 is sufficient?
  - only if RWS = 1

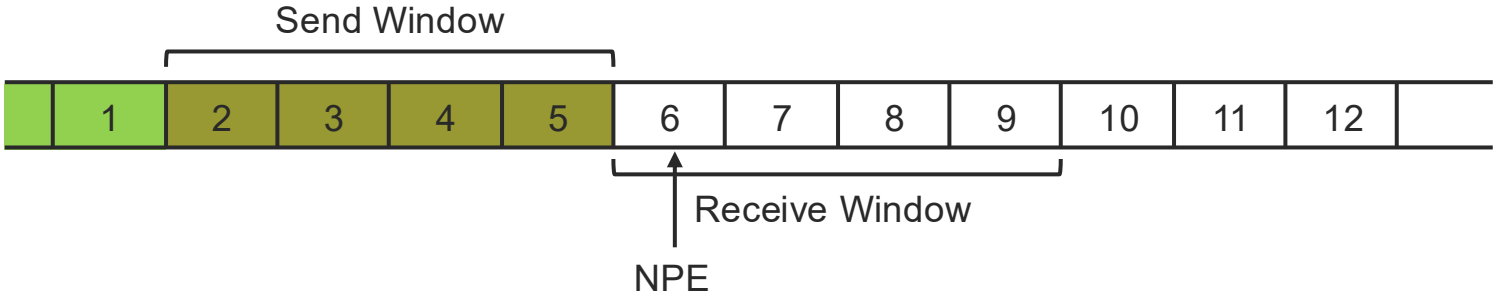
Sequence Number Space

- What if RWS = SWS?
- Example:
  - 3 bits, TotalSeqNum = 8
  - RWS = SWS = 7
  - Sender transmits 0–6
  - All arrive, but ACKs are lost
  - Sender retransmits
  - What does the receiver expect to receive?



Sequence Number Space

- A different view
  - Sender transmits full SWS
  - Receiver expects full RWS
  - Consider extreme cases
- Correctness condition:
  - TotalSeqNum ≥ SWS + RWS
    - E.g., if SWS = RWS = 8, need at least 16 sequence numbers (4 bits)
  - Alternates between two halves of sequence number space

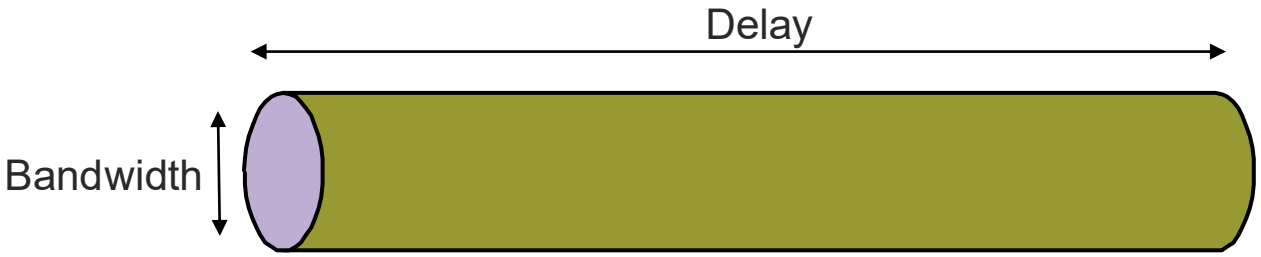


Window Sizes

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

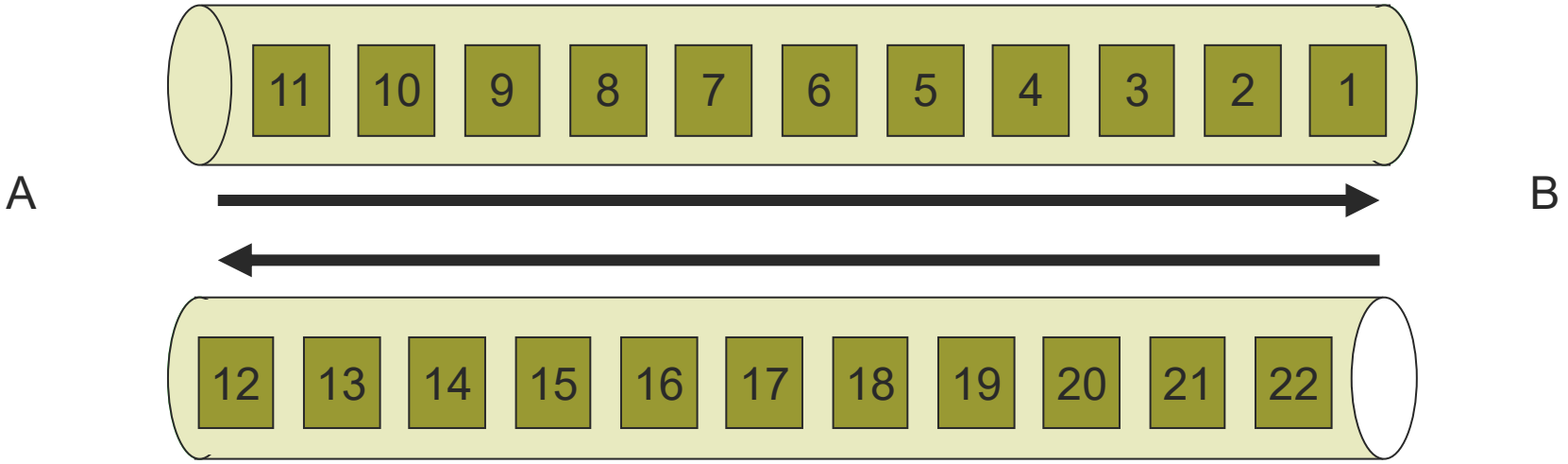
Bandwidth-Delay Product: Revisited

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



Bandwidth x Delay 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 packet sent (LPS)
  - next packet expected (NPE)
  - last packet acceptable (LPA)

Sliding Window Protocol Code Example

- Constants
  - Send/receive window size (SWS/RWS)
  - Total number of sequence numbers (TOTAL_SEQ_NO)
  - Packet size (PKT_SIZE, constant for simplicity)

Sliding Window Protocol Code Example

- Data structures
  - Next packet expected (an integer)
  - One packet buffer for each entry in receive window
  - One presence bit for each entry
- Receive window cycles (carefully) through
  - Sequence number for space efficiency

Sliding Window Protocol Code Example

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

char buf[RWS][PKT_SIZE];        /* RWS packet buffers       */
int present[RWS];               /* are packet buffers full? */
                                /*   (initialized to 0's)   */
int NPE = 0;                    /* next packet expected     */

extern void send_ack (int seq_no);          /* for sender */
extern void pass_to_app (char* data); /* for receiving app */

void recv_packet (char* data, int seq_no);   /* our focus */
```


Sliding Window Protocol Code Example

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

    /* Map sequence numbers NPE...predecessor_of_NPE
       into 0...TOTAL_SEQ_NO - 1, then see if seq_no
       falls within the receive window. */
    if (seq_no - NPE < RWS) {
                      /* packets 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]) {  /* new packet; not dup */
    present[idx] = 1; /* mark received       */
    memcpy (buf[idx], data, PKT_SIZE);
                      /* copy data into buf  */
```


Sliding Window Protocol Code Example

```
/* Got a new packet; pass as many consecutive
   packets as possible up to host */
for (i = 0; i < RWS; i++) {
    int j = (i + NPE) % RWS;

    /* first missing packet becomes NPE */
    /* after this loop terminates */
    if (!present[j]) break;

    /* packet is present – send it up! */
    pass_to_app (buf[j]);
    present[j] = 0; /* Mark buffer empty. */
}
/* Advance NPE to first missing packet. */
NPE = NPE + i;
}
```


Sliding Window Protocol Code Example

```
        }

        /* upon any received packet, send an ACK for */
        /* predecessor_of_NPE                        */
        send_ack (NPE - 1);
}
```


Sliding Window Protocol Correctness

- Claim
  - A sliding window protocol leads to reliable, in-order delivery of all packets
- Assumptions
  - Packets can be lost
  - Packets can be corrupted with detectable errors
  - Packets can be delayed an arbitrarily finite amount of time
  - All sequence numbers are different (infinite sequence space)
- Are these assumptions adequate?
  - Liveness: Any given packet is received without errors after a finite number of retransmissions (i.e., eventual delivery)

Sliding Window Protocol Correctness

- Proof sketch:
  - Use induction on packet *k*
    - Assume that packets 1,...,*k* are eventually delivered and in order
  - For packet *k*+1
    - Liveness: packet *k*+1 is eventually received
    - Sliding window protocol does not deliver packet *k*+1 before packet *k*
- What about finite sequence number space?
  - As long as it provides enough unique numbers
  - No packet can remain in the network longer than one full wraparound cycle of the sequence space

Alternative: 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