ARQ Mechanisms

CSC/ECE 570 – Fall 2024, Section 002

Need for ARQ

- Re-transmissions are needed sometimes
- Flow Control
  - Sender swamps receiver, must re-send
- Error Control
  - Error recovery after detection
  - Only non-erroneous copy at the sender
  - But– sender needs to know that

Retransmission Strategies

- *A*utomatic *R*epeat Re*Q*uest
  - Scope - Link based, or end-to-end
- Design Goals
  - Correctness
    - Sender and receiver must agree regarding what data is successfully transmitted, within small variation
    - Frames must be delivered to higher layer in order, and must not *deadlock*
  - Efficiency
    - Throughput, fraction of time channel successfully used
- Assumptions
  - Transmissions are eventually successful

Verification of Correctness

- The (re)transmission rules are embedded into protocols
- Protocol correctness must be established
  - Correctness → ?
  - Must examine “states” of endpoints
- Different methods possible

- Formal methods for less obvious cases
  - Finite State Machine Models
  - Petri Net Models

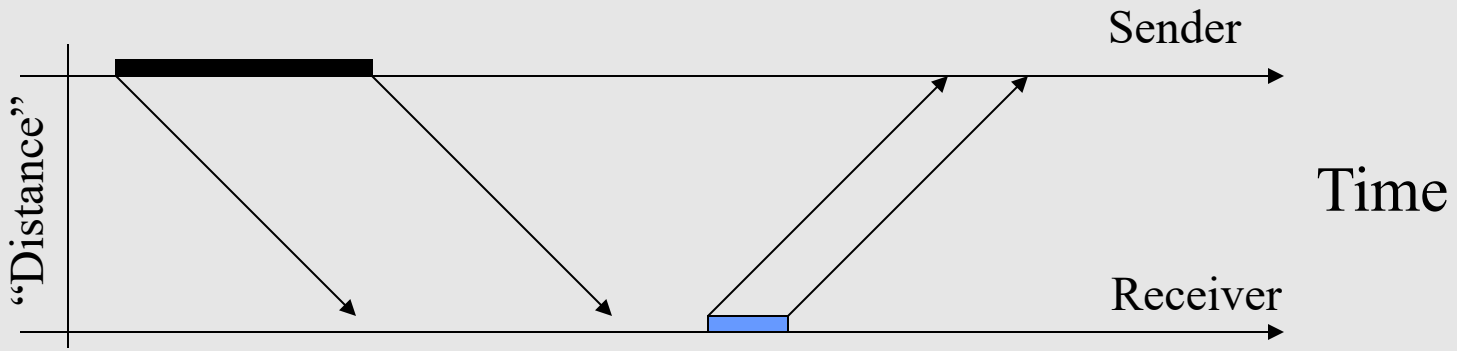
- We shall not go into formal methods in this course

Sequence numbers

- Delineated stream of bits → stream of frames
- Successive frames have different identities
- However – some may be *re*transmitted
  - It is the *same* frame
  - How to indicate
- Identity original transmission or retransmission
- Straightforward approach – sequence numbers
  - Distinguish the confusion state
- Annoying detail – how high must numbers go?
  - Finite number of bits to store such numbers
  - Very important practically !!

Timing Diagrams - Again

- Diagrams showing sequence of events involved
- Transmission of frame (start and end)
- Reception, similarly
- Delays – four kinds (focus on one link)
  - Queueing
  - Transmission
  - Propagation
  - Processing



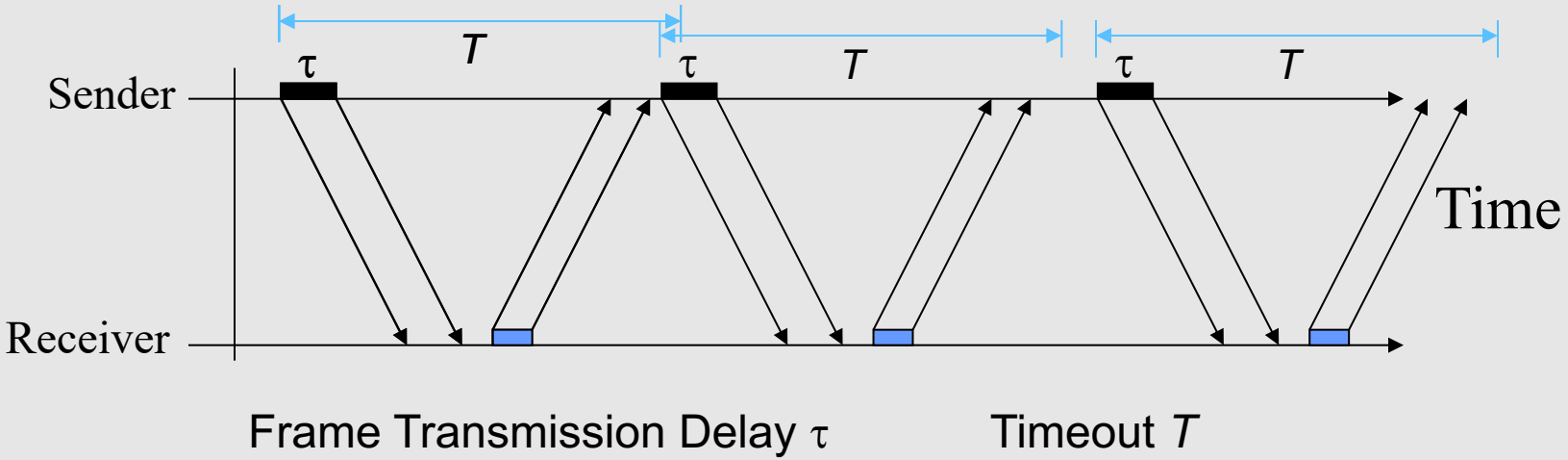
ARQ Protocols

- Stop-And-Wait (SWP)
  - What name says
  - Ignoring ACK sequence creates vulnerability to unbounded delay
- Go-Back-N (GBN)
  - Increase utilization
  - “Sliding window”
- Selective Repeat Protocol (SRP)
  - Increase efficiency further
  - “Fill up holes”

Stop-and-Wait (No ACK SN)

- Simplest scheme
- Sender assumes acknowledgement (ACK) delay bound *T*
  - Implicit request for retransmission - "timeout"
- Property: Only one outstanding frame at any time
  - Sequence numbers 0 (previous) and 1 (new) suffice
  - *General principle* - one more sequence number than maximum possible outstanding frames
  - General issue - receiver has choice in numbering ACKs
    - "I received upto frame *n*" (accumulative ACK)
    - "I received until before frame *n*"
    - Must adopt some particular convention (protocol)

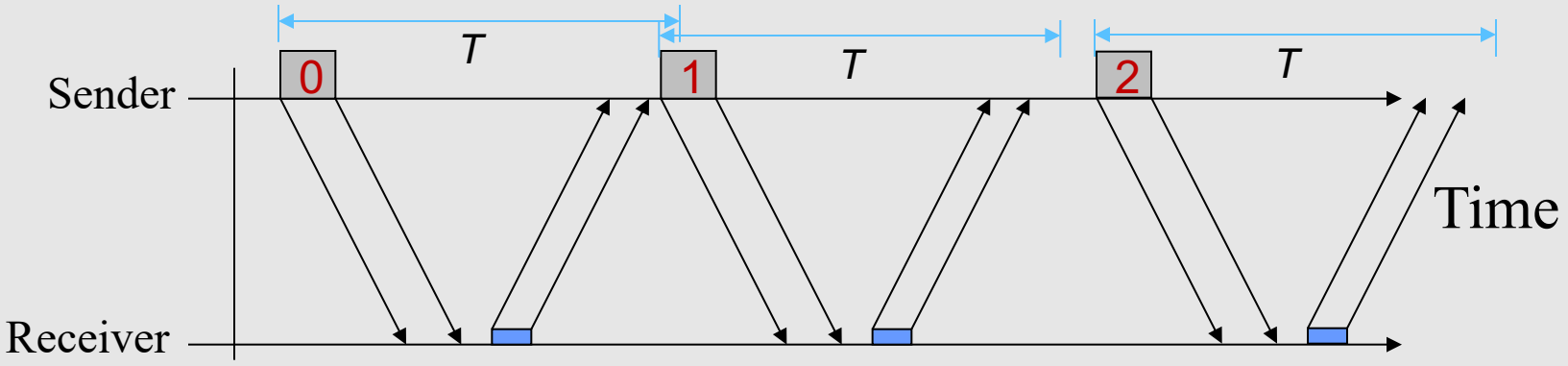
SWP Operation



Frame Transmission Delay τ Timeout T

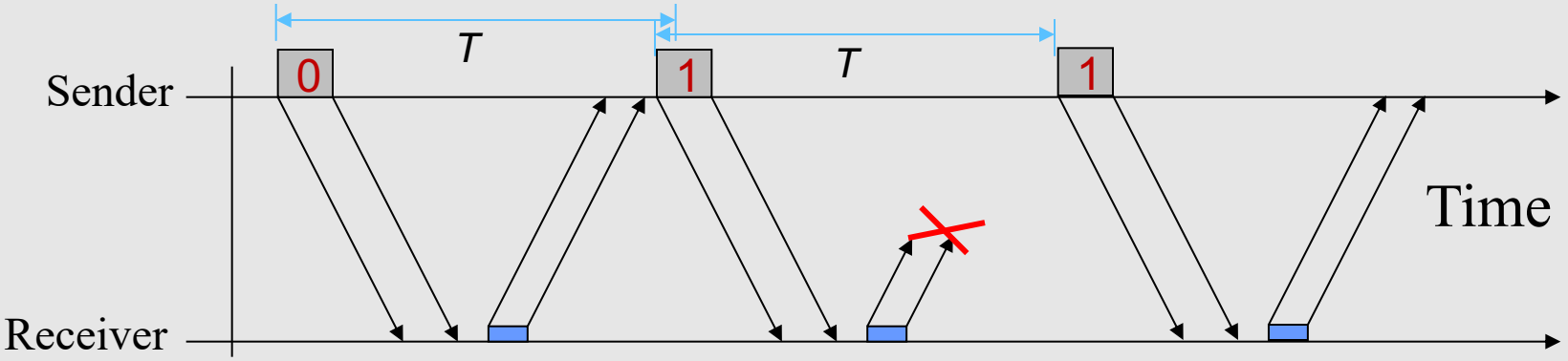
If packet is lost, timeout will occur, prompting retransmission
If ACK is lost, … ?
If ACK is delayed beyond T, … ?

SWP Operation



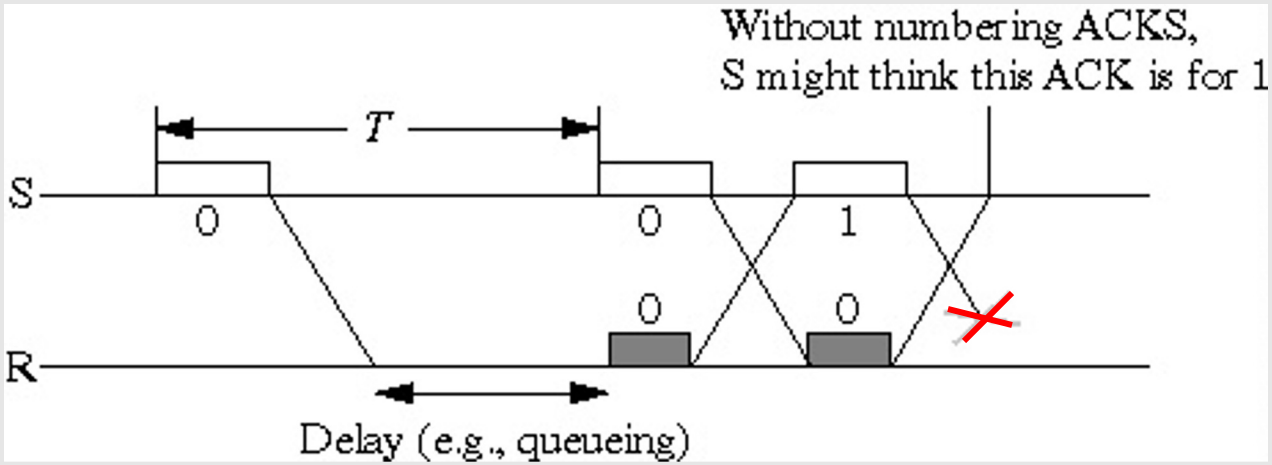
Frame Transmission Delay τ Timeout T

If packet is lost, timeout will occur, prompting retransmission
If ACK is lost, … ? -- retransmission
If ACK is delayed beyond T, … ? – already retransmit, go to send the next one



Use of Sequence Numbers

- Acknowledgements are sequence numbered
  - If eliminated, sender would be "guessing" at what frame is ACK'ed
- In general, we will mean this version when we speak of SWP

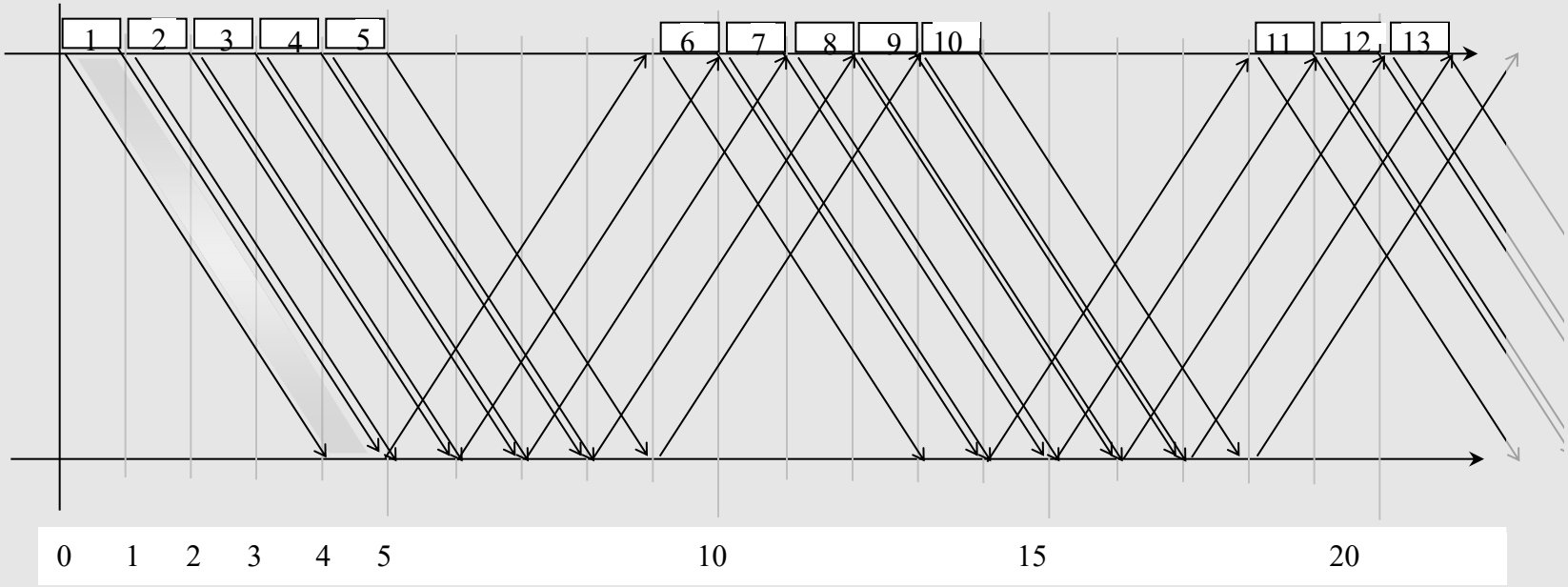


Go Back N (Sliding Window)

- Why not filling up the waiting time?
  - Sender keeps sending frames during waiting time: "Pipelining"
  - *N* is number of frames that can be transmitted
- Gain: Higher throughput in nominal operation (no errors/loss)
- Risk: Losing a frame makes all succeeding ones useless
- Discipline: Receiver only accepts in-order frame
  - At any time, is prepared to accept only the "right" frame
- Now, may need to retransmit multiple frames
  - Sender needs a buffer
    - A "window" in the sequence of frames sent
  - Limited to *N* frames, or *W* (< *N*) if only *W* buffer available
- Need at least *W*+1 sequence numbers
  - ACK is normally numbered with "next" frame
- Can use Negative Acknowledgments (NAKs) (Advantages?)

GBN Nominal Operation

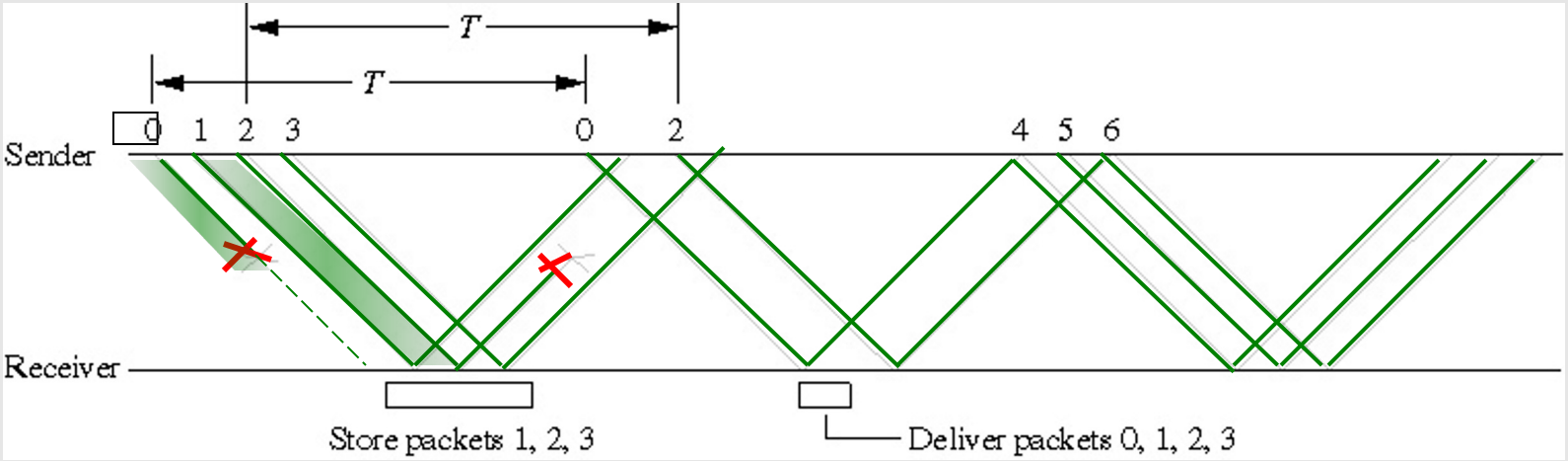
Slide window



What if some packet is lost?

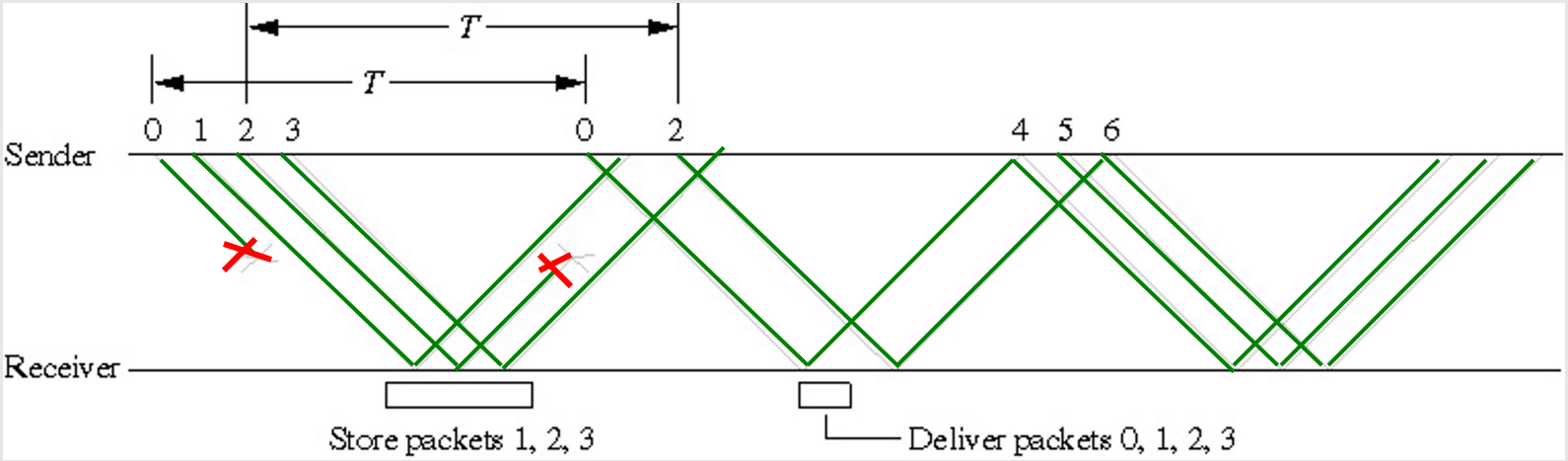
Selective Repeat Protocol (SRP)

- Why retransmit packets which made it?
- ACK for a packet *does not* mean previous ones received
  - ACK is numbered for received frame
- Same window of *W* (two windows)
  - Now receiver needs buffer also



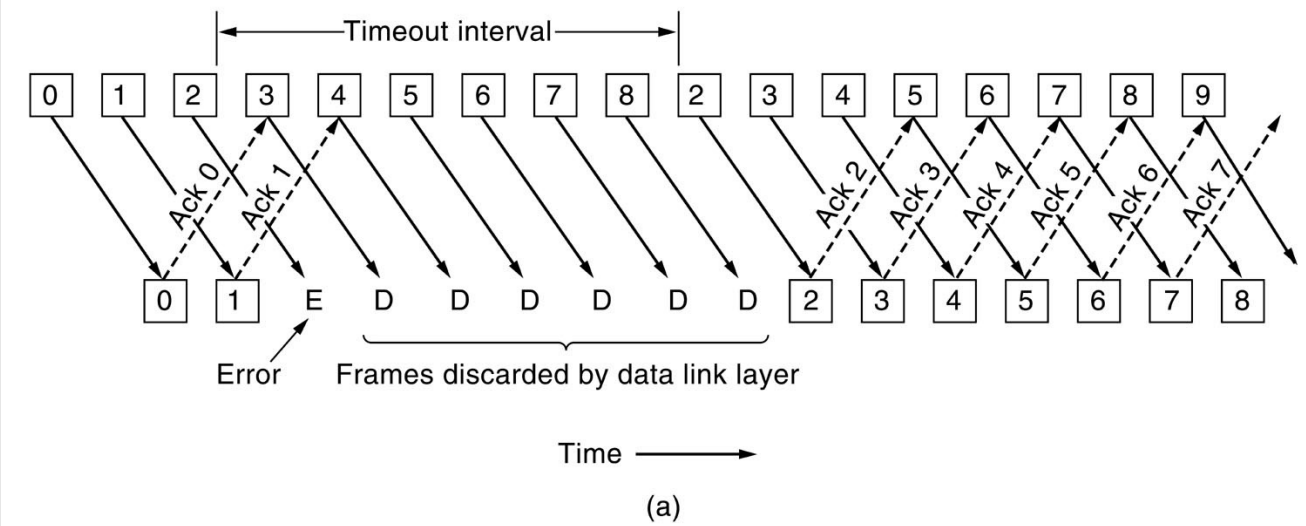
Selective Repeat

- Sender transmits packets 0 through *W*-1
- *Each* ACK is remembered
- At any time, sender can send W packets after *n*, if *all* packets 0 through *n* have been ACK'ed
- *Each* packet starts a retransmission timer

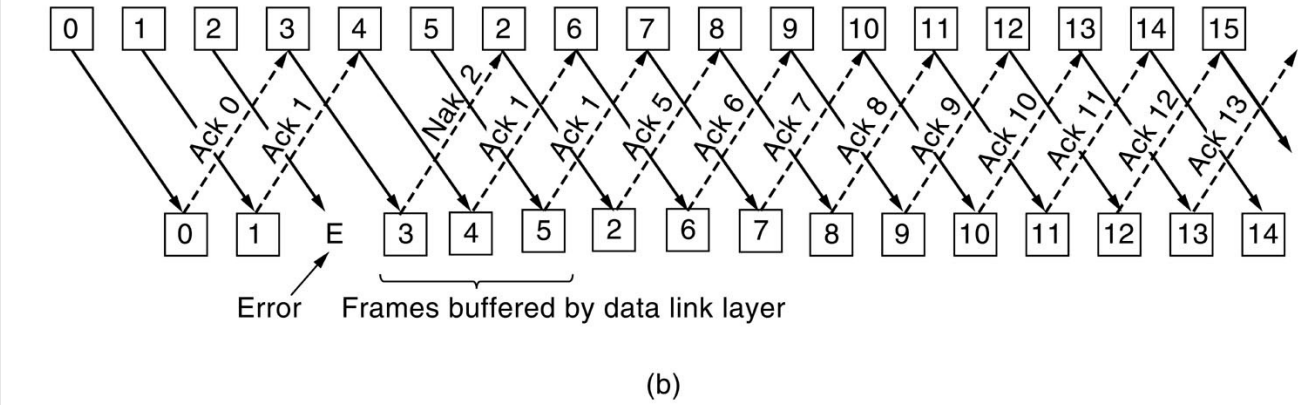


Examples: GBN vs. SRP

GBN

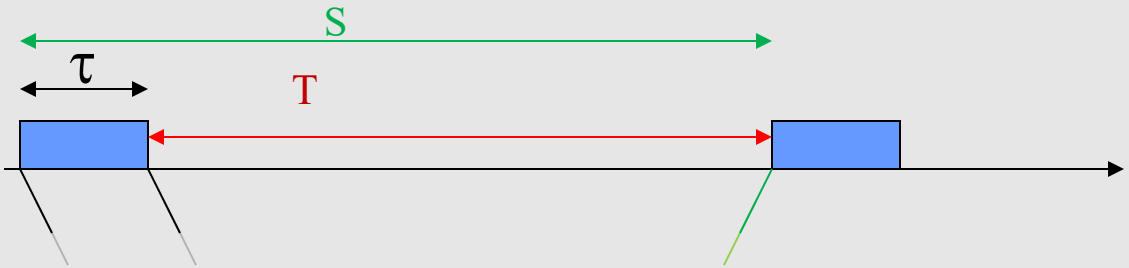


SRP



Quantitative Analyses

SWP Efficiency



- Assume sender always has frames to send
- What is the best throughput (no losses, errors)?
  - S = Time from beginning of frame transmission till ACK received
  - Simplifying assumption:
    - ACK is received just before timeout, S = τ + T (total time)
  - Efficiency = Fraction of time channel is busy = τ / S
  - Throughput: 1/S = 1/(τ + T) frames/second

- Now consider the case with errors
  - Model the uncertainty of a successful transmission with probability
  - A transmission is successful with probability *p*
    - True for any packet => memoryless process

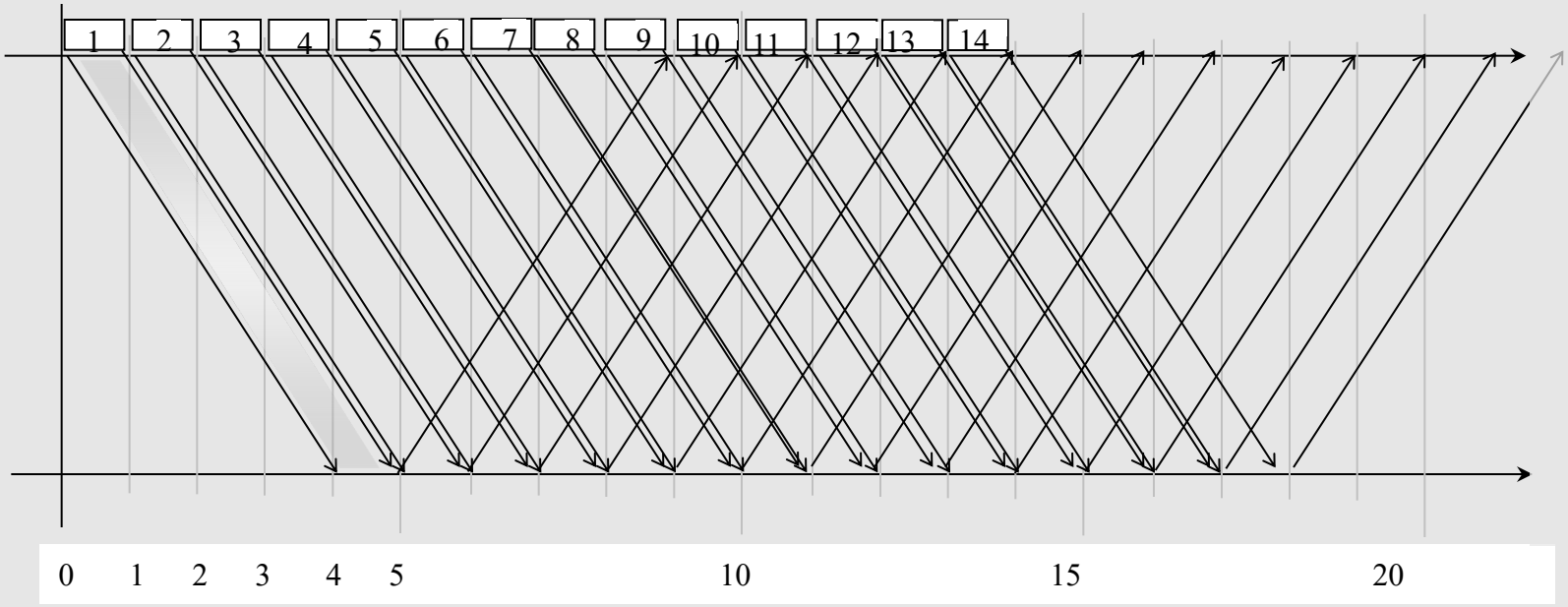
SWP Efficiency

- Approach: find the expected time to transmit any given frame successfully
  - First find the probability distribution (pmf)
  - Probability that exactly 1 transmission is required = ??? p
  - Probability that exactly 2 transmissions are required = ??? $(1-p)p$
  - Probability that exactly k transmissions are required = ??? $(1-p)^{k-1}p$
- Expected time before the frame is successfully transferred = ??? $\Sigma_{(1,n)}\, k*S*(1-p)^{k-1}p$

  $S*[1-(1-p)^n]/p - S*n*(1-p)^{n-1}$
- Throughput = ??? $1/(\tau + T)*[1/([1-(1-p)^n]/p - n*(1-p)^{n-1})]$

GBN Nominal Operation

- "Full" window - enough to keep transmitting through ACK



- Key observation
  - For SWP, Success takes time τ + T, Failure ALSO takes time τ + T
  - For GBN
    - Success takes time = ? (W*τ + T) for W frames (τ + T/W for 1 frame)
    - Failure takes *same* amount of time as before
- Hence throughput expression as before

GBN Operation

- Approach: find the expected time to transmit any given frame successfully
  - First find the probability distribution (pmf)
  - Probability that exactly 1 transmission is required = ???
  - Probability that exactly 2 transmissions are required = ???
  - Probability that exactly k transmissions are required = ???
- Expected time before the frame is successfully transferred = ???

$$S*[1-(1-p)^n]/p - S*n*(1-p)^{n-1} \qquad S = W*\tau + T$$

- Throughput = ???

$$W/(W*\tau + T)* [1/([1-(1-p)^n]/p - n*(1-p)^{n-1})]$$

How To Set the Window Size?

- Idea: to fill the pipe entirely
  - w: window size (to set)
  - B: bandwidth (bps) divided by frame size (bits-per-frame)
    - Expressed in terms of frame rate, or fps (not fps in video gaming :)
  - D: one-way propagation time/delay
  - BD: bandwidth-delay product
  - Upper bound on link utilization:

$$\text{link utilization} \leq \frac{w}{1 + 2BD}$$

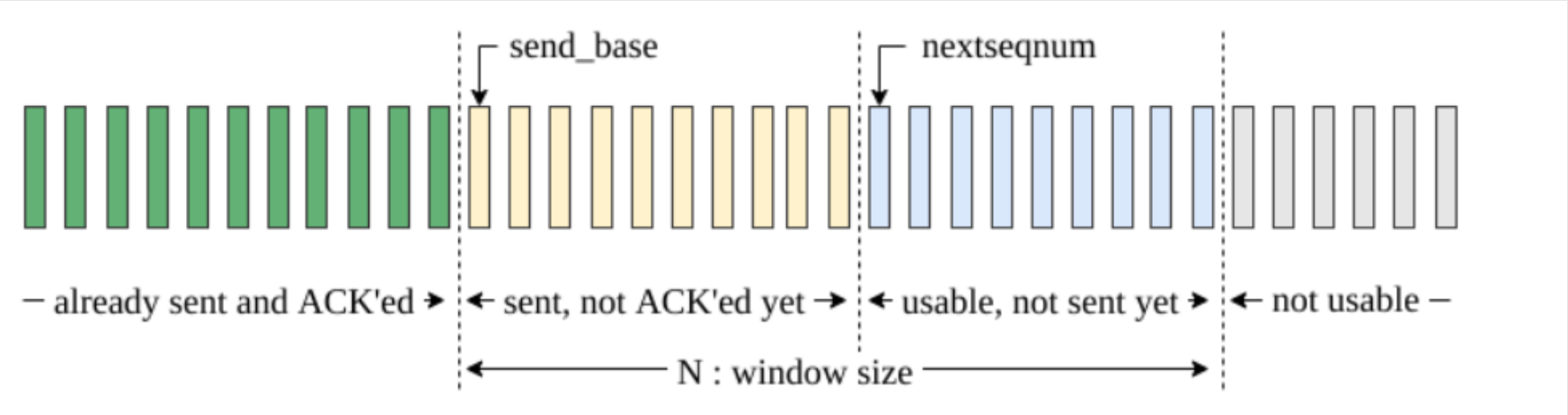
  - w ~ 1 + 2BD (how many frames in total)

Implementation of GBN

GBN

- The sliding window (pipelined) protocols achieve utilization of network bandwidth
- not requiring the sender to wait for an acknowledgment before sending another frame.
- Just require a simple receiver
- Sequence number, Window size

GBN


send_base
nextseqnum
already sent and ACK'ed
sent, not ACK'ed yet
usable, not sent yet
not usable
N : window size

GBN Sender

Algorithm 1: Go-Back-N Sender

```
Function Sender is
    send_base ← 0;
    nextseqnum ← 0;
    while True do
        if nextseqnum < send_base + N then
            send packet nextseqnum;
            nextseqnum ← nextseqnum + 1;
        end
        if receive ACK n then
            send_base ← n + 1;
            if send_base == nextseqnum then
                stop timer;
            else
                start timer;
            end
        end
        if timeout then
            start timer;
            send packet send_base;
            send packet send_base + 1;
            ...
            send packet nextseqnum − 1;
        end
    end
end
```

GBN Receiver

Algorithm 2: Go-Back-N Receiver

```
function Receiver is
    nextseqnum ← 0;
    while True do
        if A packet is received then
            if The received packet is not corrupted and
             sequence_number == nextseqnum then
                deliver the data to the upper layer;
                send ACK nextseqnum;
                nextseqnum ← nextseqnum + 1;
            else
                /* If the packet is corrupted or out of
                   order, simply drop it                    */
                send ACK nextseqnum − 1;
            end
        else
    end
end
```

ARQ Sequence # 1

ARQ Sequence Number 1

Why sequence number is needed in the data packet in ARQ?

A	Because receiver needs to distinguish between a new packet and a retransmission packet
B	Because sender needs to distinguish between a new packet and a retransmission packet
C	Because receiver needs to distinguish between ACKs of a new packet and a retransmission packet
D	Because sender needs to distinguish between ACKs of a new packet and a retransmission packet

ARQ Sequence #2

Why sequence number is needed in the ACK in ARQ?

A	Because receiver needs to distinguish between a new packet and a retransmission packet
B	Because sender needs to distinguish between a new packet and a retransmission packet
C	Because receiver needs to distinguish between ACKs of a new packet and a retransmission packet
D	Because sender needs to distinguish between ACKs of a new packet and a retransmission packet

Summary

- ARQ protocols contribute to error and flow control at the DLC layer
- We are interested in
  - Correctness
  - Efficiency
- Even simple protocols can be complicated
- Must know
  - How to draw timing diagrams
  - How to argue correctness informally
  - How to apply probability to obtain efficiency/throughput
  - How to “run the protocol in your mind”

Summary

- ARQ protocols contribute to error and flow control at the DLC layer
- We are interested in
  - Correctness
  - Efficiency
- Even simple protocols can be complicated
- Must know
  - How to draw timing diagrams
  - How to argue correctness informally
  - How to apply probability to obtain efficiency/throughput
  - How to “run the protocol in your mind”

Problem for MAC

- Can we transfer data reliably between two point-to-point nodes?

- How to make multiple nodes talks in a network?