# Lab 2: Data Link Layer and ARQ Protocols

ECE 358

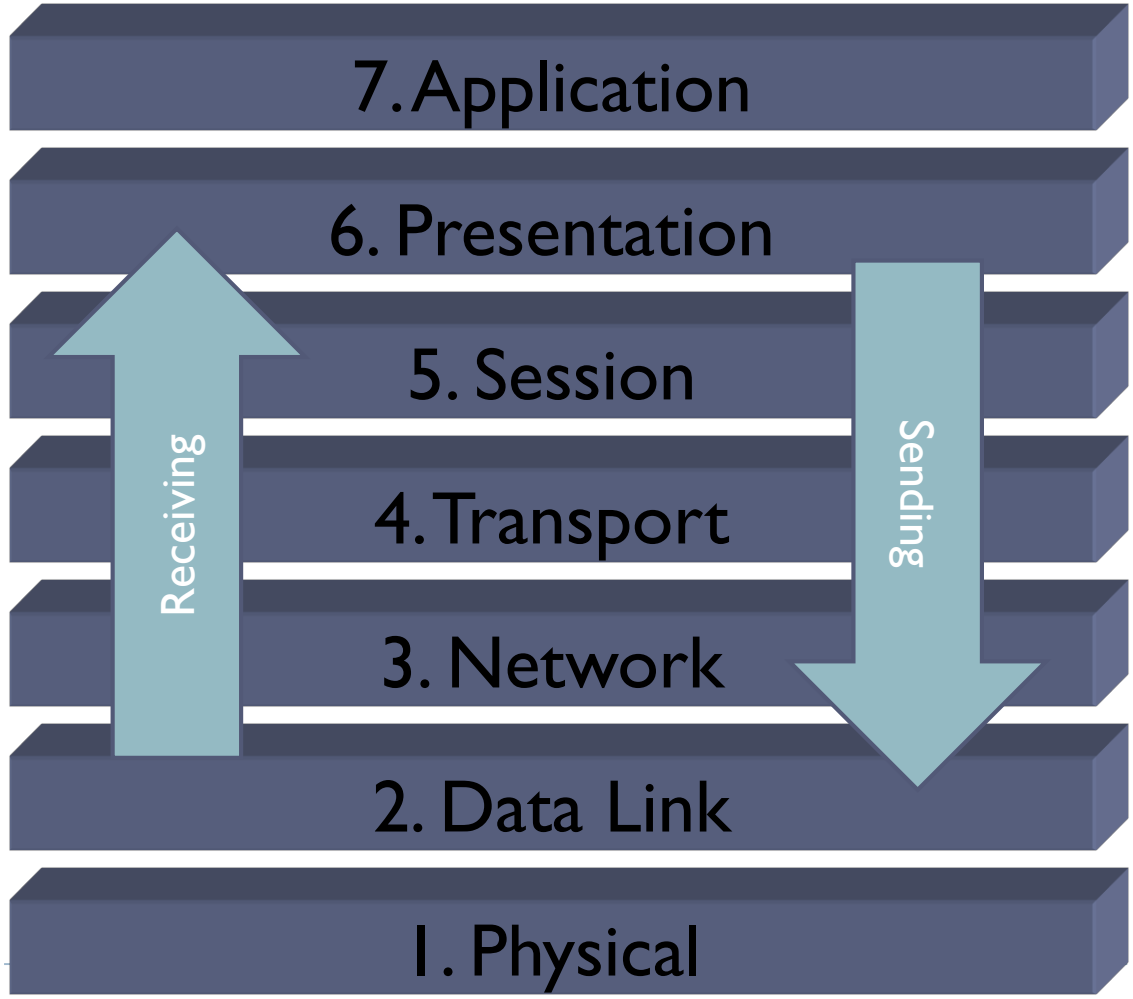
Presented by Bernie Roehl

Based on original slides by Albert Wasef

# OSI Model

- The concept of layered architecture is essential to the design of a communication networks to be capable of sharing information.
- There is a need for a standard that governs how this information is formatted, transmitted, received and verified to enable information to be shared openly, even when dealing with dissimilar networks.
- The Open Systems Interconnection (OSI) model is an ISO (International Standards Organization) standard that covers all aspects of network communications.
- The OSI model is a layered framework for the design of network systems that allows communication between all types of computer systems.

# OSI Model
# The Layered Approach to Communication




All

People

Seem

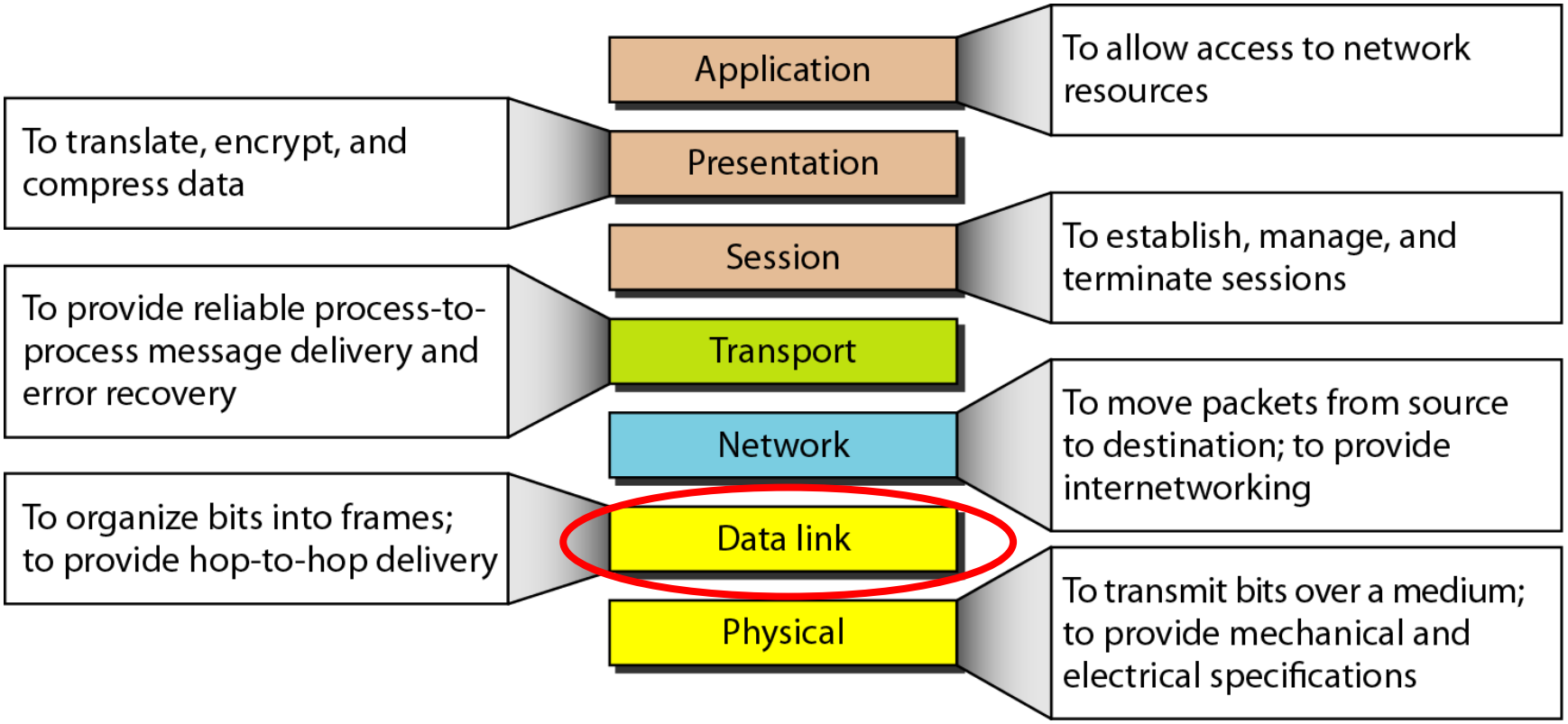
To

Need

Data

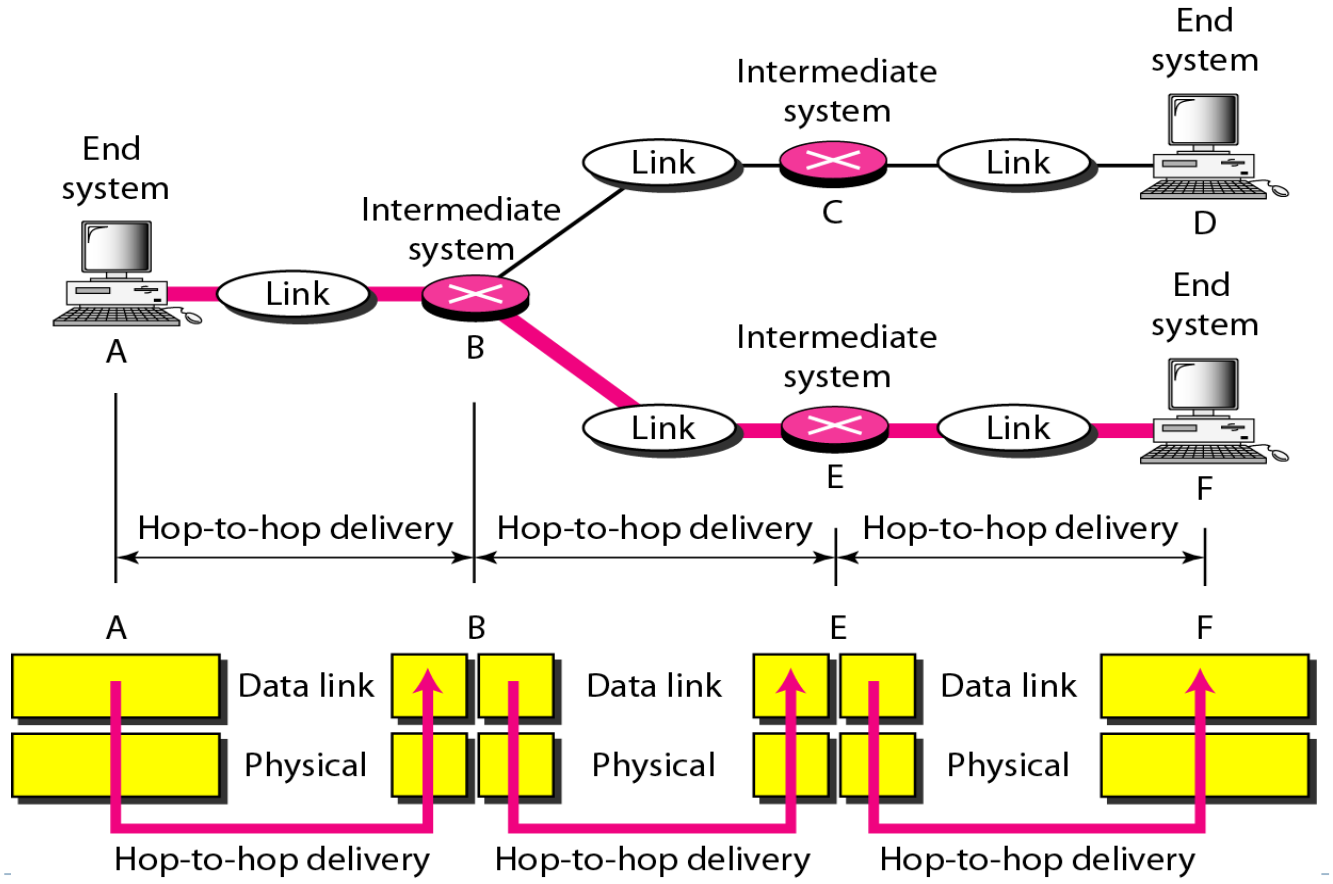
Processing

# OSI Model
# The Layered Approach to Communication



# Data Link Layer

- The data link layer is responsible for moving packets from one hop (node) to the next.



# Data Link Layer

- Layer 2 provides reliable point-to-point communication service
  - Forward Error Correction (FEC)
  - Automatic Repeat Request (ARQ)
- In this lab, we will study ARQ.

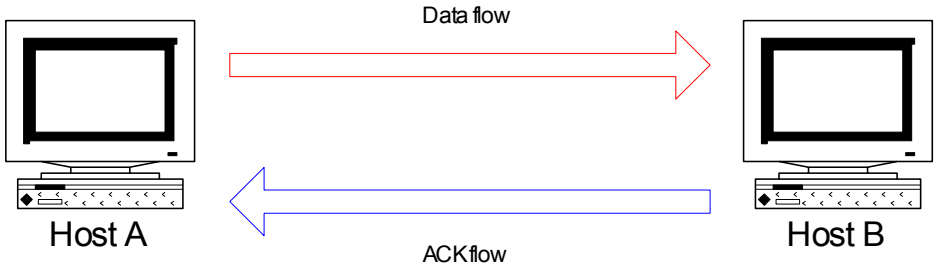
- The basic concept of ARQ protocols is to detect packets with errors at the receiving end and request the sender to retransmit the erroneous packets.

- The performance metric of interest is the efficiency:

how much of the channel bandwidth is wasted by unnecessary waiting and retransmission

# Automatic Repeat Request (ARQ) Protocols

- We are going to consider two ARQ protocols
  - *Alternating Bit Protocol (ABP) – this session*
  - *Go Back N (GBN) – next session*
  - We will assume a bidirectional link between A and B



# Alternating Bit Protocol (ABP) -- Sender

- We transmit a packet from layer 3 with serial number SN
- We then wait for an ACK that says the receiver is ready for packet SN+1 (modulo 2)
- When we get that ACK, we increment the SN (again, modulo 2) and transmit the next packet
- If we don't get an uncorrupted ACK before a certain timeout, we re-send the packet (with the same serial number)
- If we get a corrupted ACK, or an out-of-sequence ACK, we can do one of the following:
  1. Ignore it and wait for the timeout
  2. Resend the current packet

# Alternating Bit Protocol -- Receiver

- We keep track of the next SN we're expecting (initially zero)
- When an uncorrupted packet arrives whose SN matches, we deliver it to layer 3 and then send an ACK containing the next SN (modulo 2) to tell the sender we're ready for the next packet
- If we receive a corrupted data packet, or a packet with an unexpected SN, we resend the most recent ACK

# ABP Sender in Detail

- Buffer space for 1 packet
- Sequence numbers alternate between 0 and 1
- Two counters:
  - SN: (Initialized to 0)
  - NEXT_EXPECTED_ACK which is always (SN +1) % 2

# ABP Sender in Detail (continued)

1) Get a new packet from Layer 3

2) Send the packet with sequence number SN at time *t*

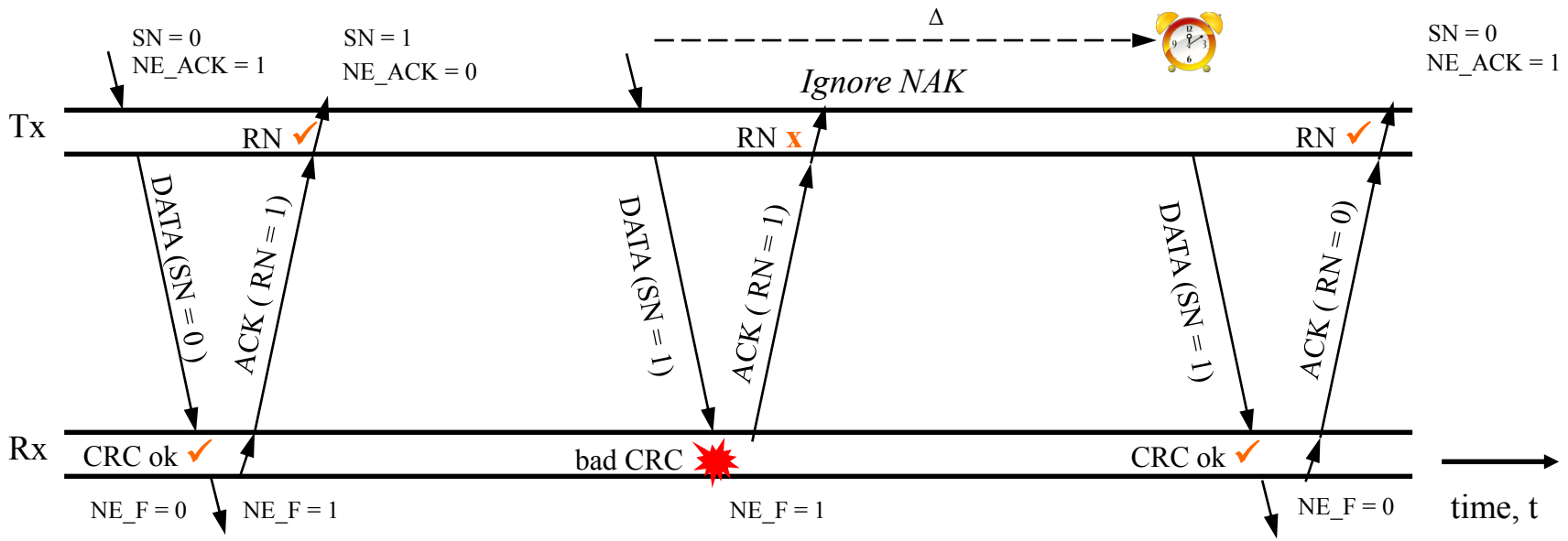
- We also *reset* the time-out event to occur at $t + \Delta$

3) Wait for an ACK or a TIMEOUT event

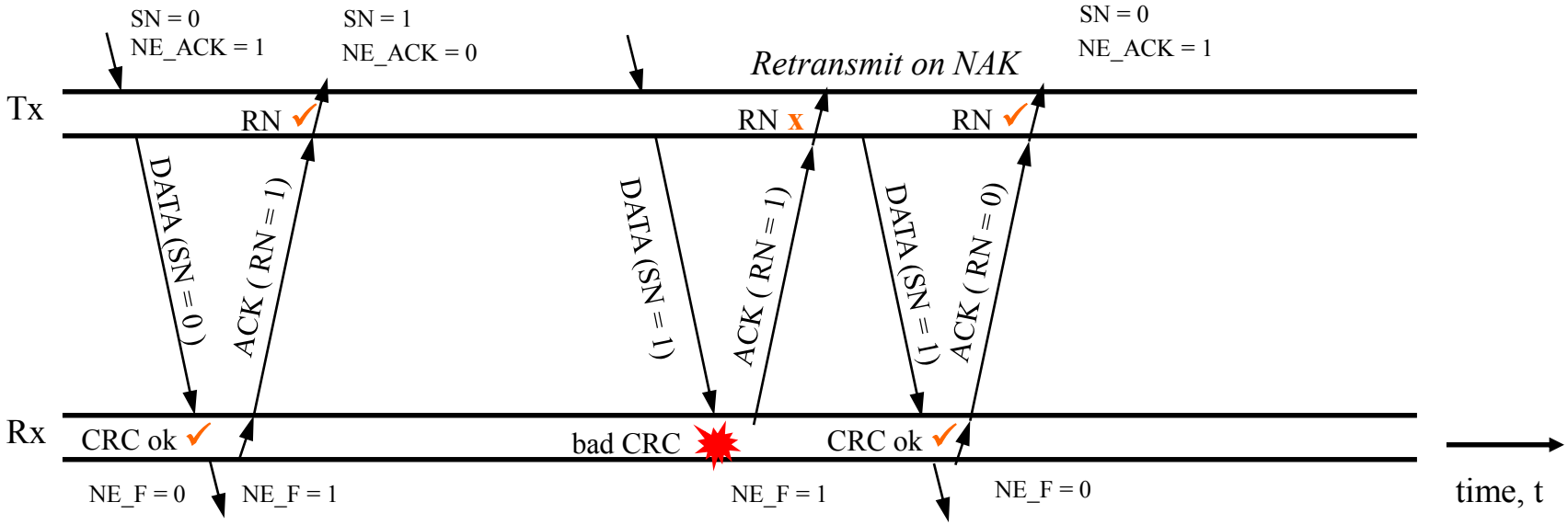
- If it's a TIMEOUT, go to (2), i.e. retransmit
- If ACK
  - If ACK is not corrupted and has NEXT_EXPECTED_ACK
    - increment SN and NEXT_EXPECTED_ACK (modulo 2), go to (1)
  - If ACK is corrupted or has a mismatched SN
    - Type 1 sender (Question 1) do nothing, i.e., go to (3)
    - Type 2 sender (Question 2) go to (2), i.e. retransmit

# ABP Receiver protocol

- Initialize NEXT_EXPECTED_FRAME to zero
- Wait for a new received packet
  - If packet arrives with no error and matches NEXT_EXPECTED_FRAME:
    - deliver the packet to Layer 3
    - increment NEXT_EXPECTED_FRAME (modulo 2)
    - send an ACK with a acknowledgement number RN equal to NEXT_EXPECTED_FRAME
    - go back to waiting
  - If corrupted packet, or packet with wrong SN arrives:
    - send an ACK with same RN as we last sent
    - go back to waiting




*Question 1: ABP sender that ignores bad packets*



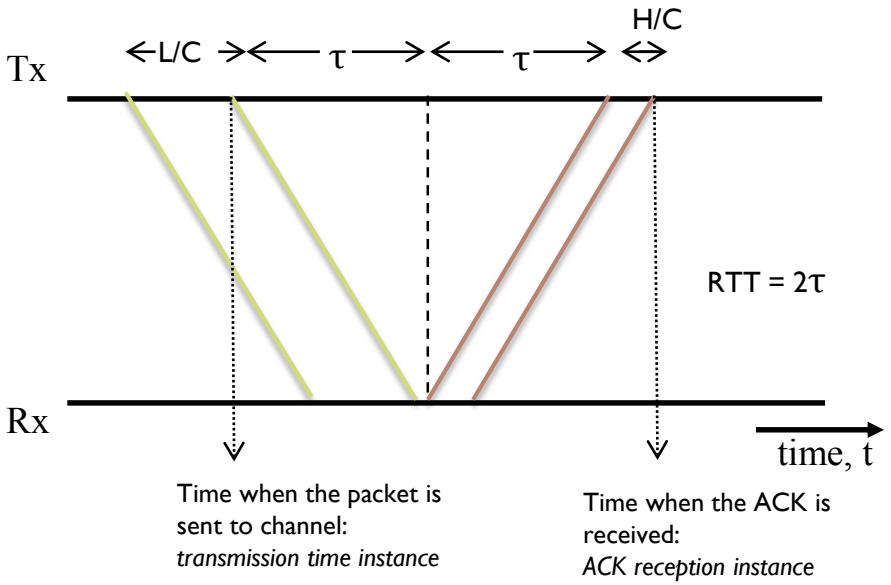

*Question 2: ABP sender that resends on error*

# Packet Corruption and Loss

- Packets can be lost, including both data packets and ACKs
- Packets can arrive, but be corrupted (i.e. have errors)
- For simulation, you are given a Bit Error Rate (BER)
- BER is the probability of a given bit being corrupted
- For each of the L bits in a packet, you generate an error with probability BER
- If number of errors == 0, deliver uncorrupted packet
- If number of errors > 5, drop packet (5 is arbitrary)
- Otherwise, deliver packet with error flag set
- In this lab, all the packets have the same length

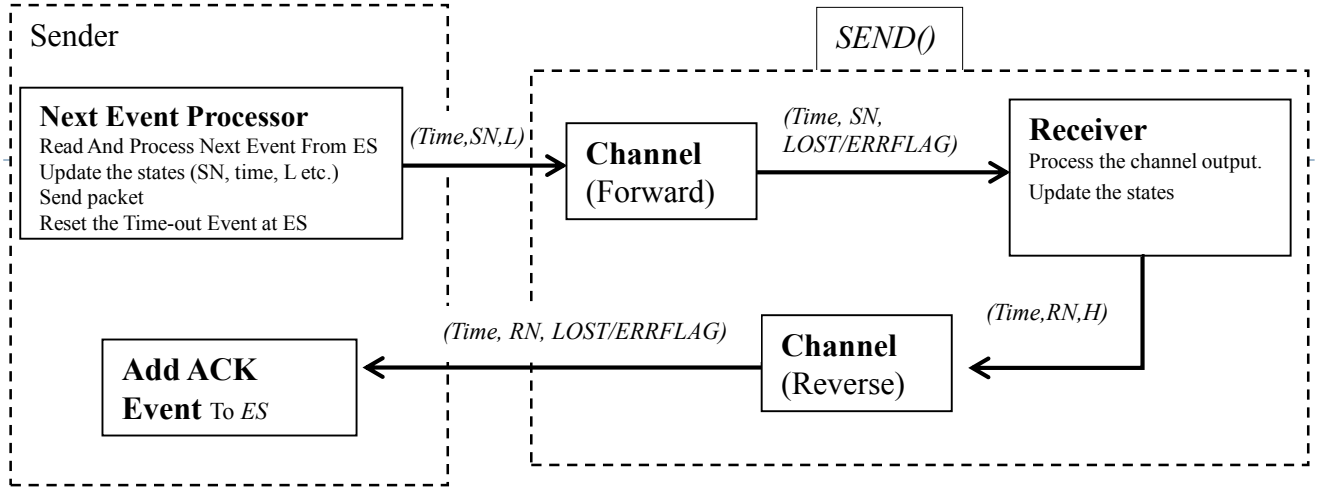
# Channel Definition and Timing

- Capacity of C bit/second
- Bit Error Rate (BER)
- Constant propagation delay τ



# DES Simulation Framework

- Event Scheduler (ES), similar to Lab 1
  - Event has the following fields
    - *type* (TIMEOUT or ACK)
    - *timestamp*
    - *SN (only if type == ACK)*
    - *error-flag (only if type == ACK)*
  - Event Scheduler (ES) stores events in sequence
    - *ES is typically a linked list of events, in order of timestamp*
    - *Useful functions that you can create:*
      - register_event(Event evt):
        - *adds an Event to the queue*
        - *inserted at the right position (based on its timestamp)*
      - dequeue(): *returns (and removes) the next event (earliest event)*




SEND()

Input: ***Time*** is the time at which the packet was sent to the channel
***SN*** is the serial number of the packet
***L*** is the length of the packet

Output: ***Time*** is the time at which the ACK was fully received at the sender
***RN*** is the SN of the next packet we're expecting
A flag representing whether either the packet or the ACK was **lost**
A flag representing whether the ACK arrived with **errors**

Simulation is initialized by sending the first packet, which will add events to the ES

Cycle begins with sender reading the ES and responding to the next event

If no packet loss, an ACK (error free or in error) event is registered in the ES

# Important notes regarding time

- The current time counter should be updated only in the following cases:
  - After transmitting a packet from the sender's buffer.
  - Reading an event from the event scheduler, i.e., set the current time equal to the event time.
- The current time should not be updated when calculating the time of the ACK, i.e., while the packet is in transition through forward channel, receiver, and reverse channel
- That is because the sender should be transmitting packets irrespective of what delay the latest transmitted packet is experiencing after it left the sender

# A note on TIMEOUT

- **There can be only one TIMEOUT event in the ES**
  - Implement a purge_time_out() function in your ES to delete any timeout event in your ES before registering a new timeout event

# Performance metric

- Throughput:
  - Number of information bits transmitted per second (i.e. not counting any headers)
- Different protocols will have different performance
- Heavily dependent on
  - channel parameters (C, BER, etc)
  - packet lengths, etc.

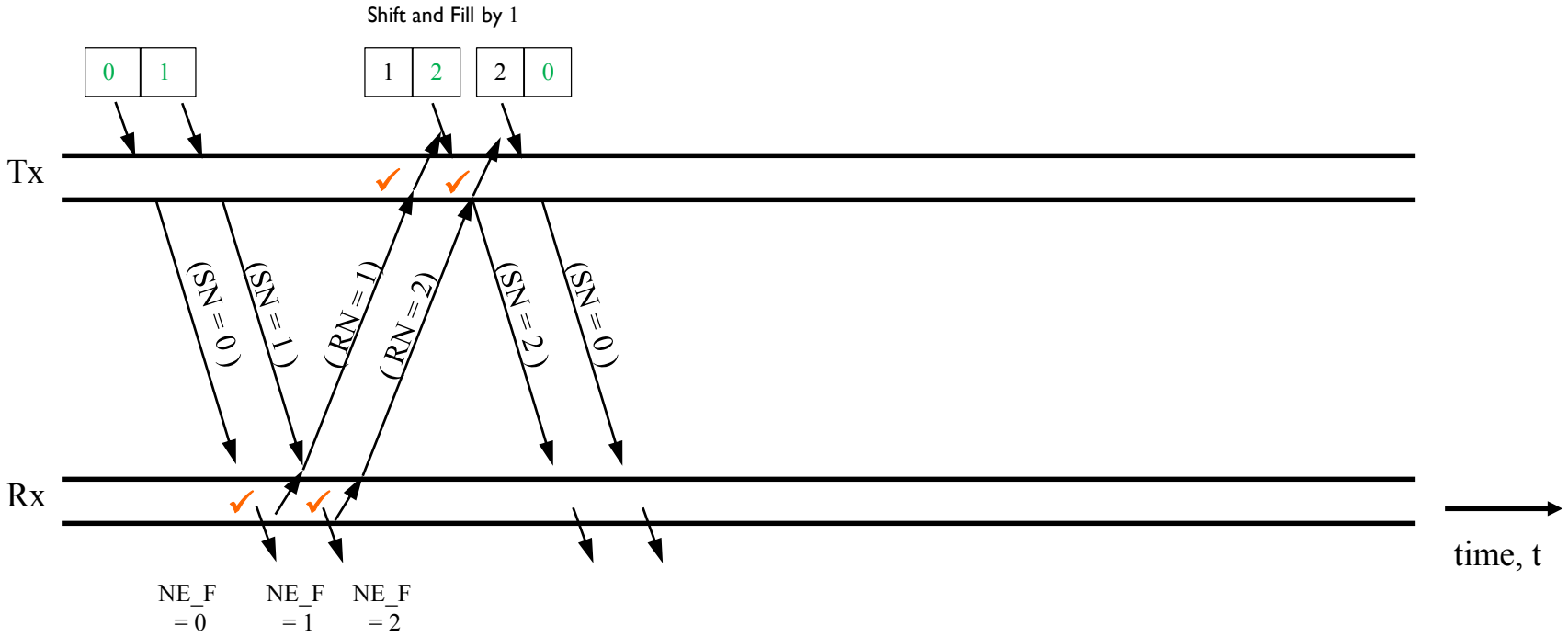
# Final words

- Read the lab manual carefully
  - Follow the *details and conventions* from the lab manual.
  - Otherwise, your results might be different from what we expect.
- Start early
- Compare your results with theoretical results (whenever they are available), to check the correctness of your code.

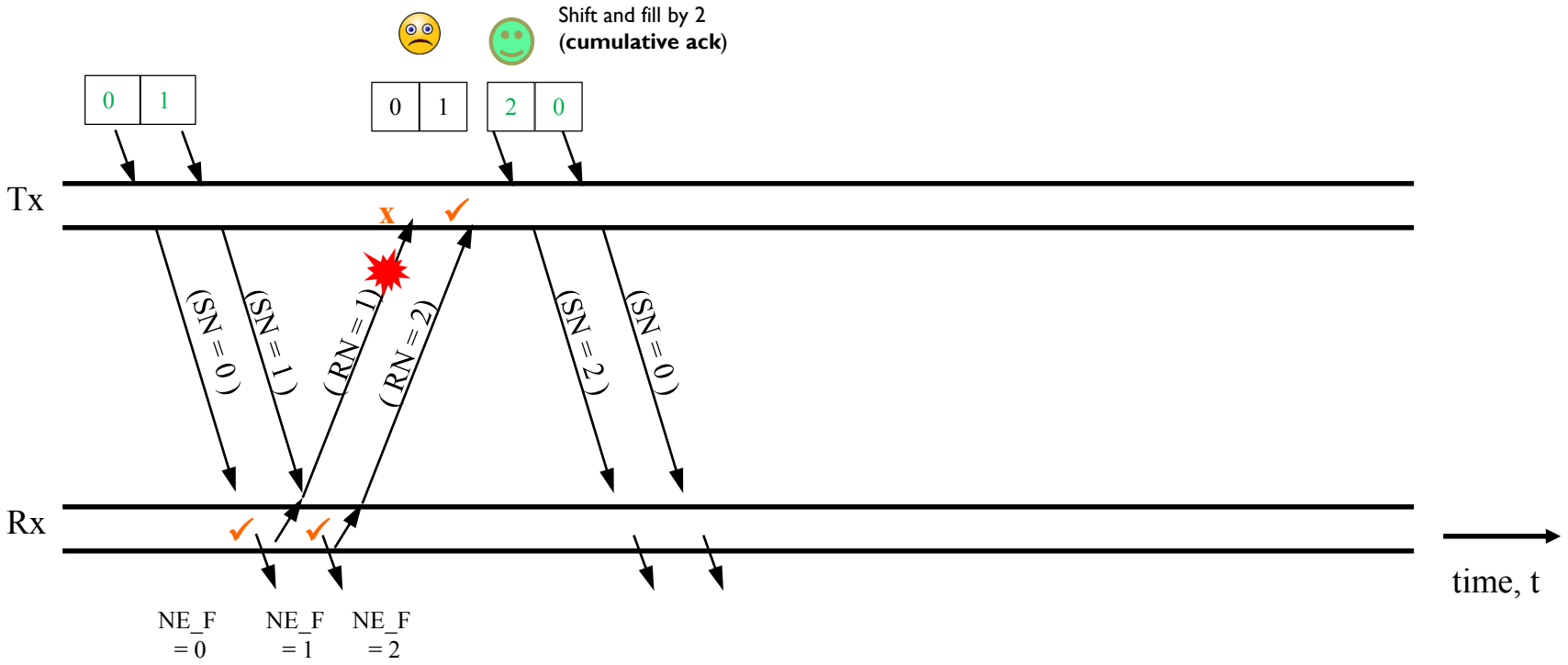
Thank you

# GBN

- Except for the sequence numbering, *the GBN receiver is exactly the same as the ABP receiver*
  - The sequence number goes 0, 1, 2, .., N before cycling back to 0
- The *channel is also of course unaffected*.
- **The Sender is quite different**
  - **Sender can send up to N packets** before the oldest packet is acknowledged.
  - Sender has a buffer space for N packets, not just one




Demonstration of GBN (N=2) when channel is error-free.




Demonstration of GBN (N=2) when ACK is lost or corrupted.

**the cumulative acknowledgement can *minimize* to some extent, the negative effect of ACK loss/errors.**

# GBN Sender Details

- Sender has a buffer for N packets.
  - P = SN[1] is the oldest packet in the buffer.
- Initially, the buffer is filled with N *new* packets.

# GBN Sender Details (continued)

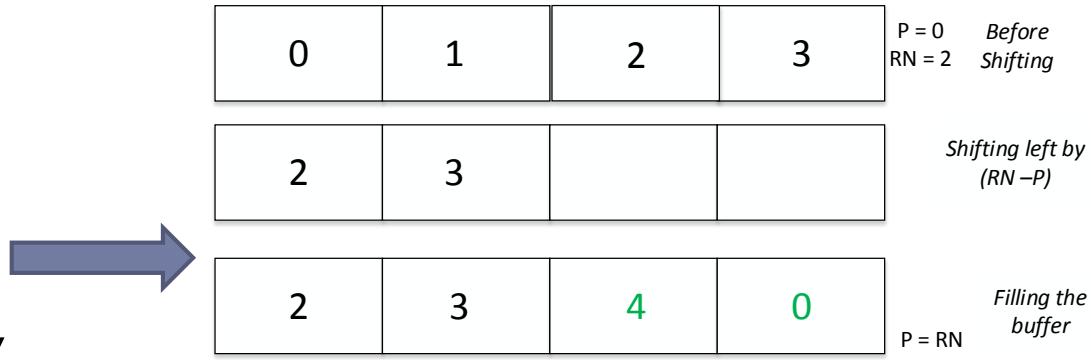
- 1) Sender transmits the packet whose turn it is to be transmitted (i.e., only one packet) out of all the packets to be transmitted in the buffer.
- 2) If the transmitted packet in step 1 is the oldest packet in the buffer, the sender registers a TIME-OUT event in the ES at time T[1] + Δ.
- 3) Compare the *current time* $t_c$ with the time of the *next event* in the ES:
  - 3.1) If $t_c$ < the time of the next event, go to step 1.
  - 3.2) If $t_c$ > the time of the next event, then you will have to process the *next event* in the ES as indicated in steps 5-7. (**Very important**).
- 4) After transmitting all the packets in the buffer, read the next event from the ES.

# Event processing: a note

- Note that a sender can transmit more than 1 packet sequentially (unlike in ABP)
- What if a time-out or an ACK event occurs in the middle of a packet transmission?
  - While the sender is sending packet *i*, starting at $t_c$, it cannot respond to events that occur during that transmission (i.e., between $t_c$ and $t_c$ + L[i]/C).
  - An event that occurred during this interval has to be assumed to have occurred at $t_c$ + L[i]/C even though it occurred before this time.

# GBN Sender Details (3)

- 5) If next event is TIME-OUT
  - All N packets in buffer must be **retransmitted** (***Go Back N***)
- 6) If next event is an ACK, 2 situations are possible:
  - *Situation 1*
    - ACK not corrupted *and* **RN is either of P + 1, P+ 2, …, P + N (modulo N+1)**
  - *Situation 2:*
    - if ACK corrupted *or*
    - if RN = P
  - If *situation 1*,




    - *The buffer is shifted by (RN – P) modulo (N+1) and filled with new packets (**Window slides**)*
    - *The new packets are sent to the receiver*

# GBN Sender Details (4)

- If *situation 2*,
  - **Ignore it**
- 7) Update time-out whenever SEND() is called

  (*only one time-out in ES at a time*)
- Implementation :
  - An additional data structure to simulate the buffer,
    - functions for *shifting* and *filling*
  - Sender's event processing details
  - Reuse the components from ABP simulator
    - *channel*, *receiver* (with a tiny modification), *ES* etc.
    - The DES framework remains the same.