CSC4200/5200 – COMPUTER NETWORKING

RELIABLE DELIVERY – PART 1

Instructor: Susmit Shannigrahi
sshannigrahi@tntech.edu

Tennessee TECH

Apps (HTTP)

Data

Apps (HTTP)

Transport (TCP/UDP)

Segments

Transport (TCP/UDP)

Network (IP)

Packets

Network (IP)

Link (Ethernet)

Frames

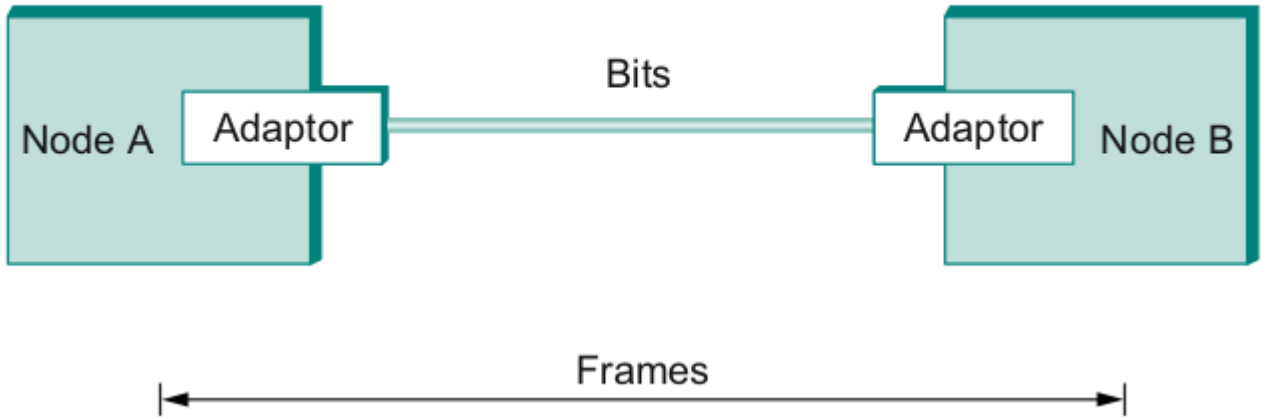
Ethernet Interface

Ethernet Interface

Link (Ethernet)

Bits (1010001)

Frames – bag of bits

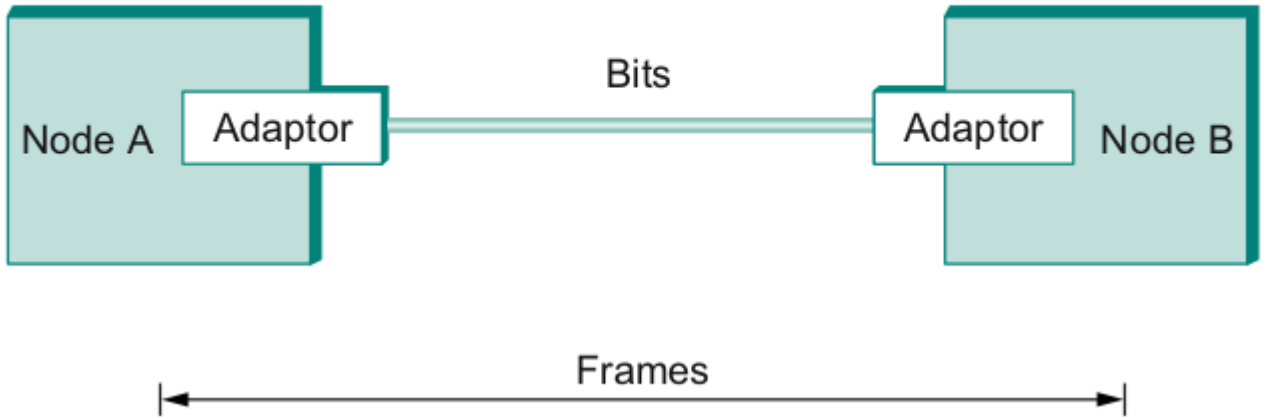


- Sending side – encapsulation, add error check bits, flow control
- Receiving side – extract frames, check for error, flow control

Reliable Delivery

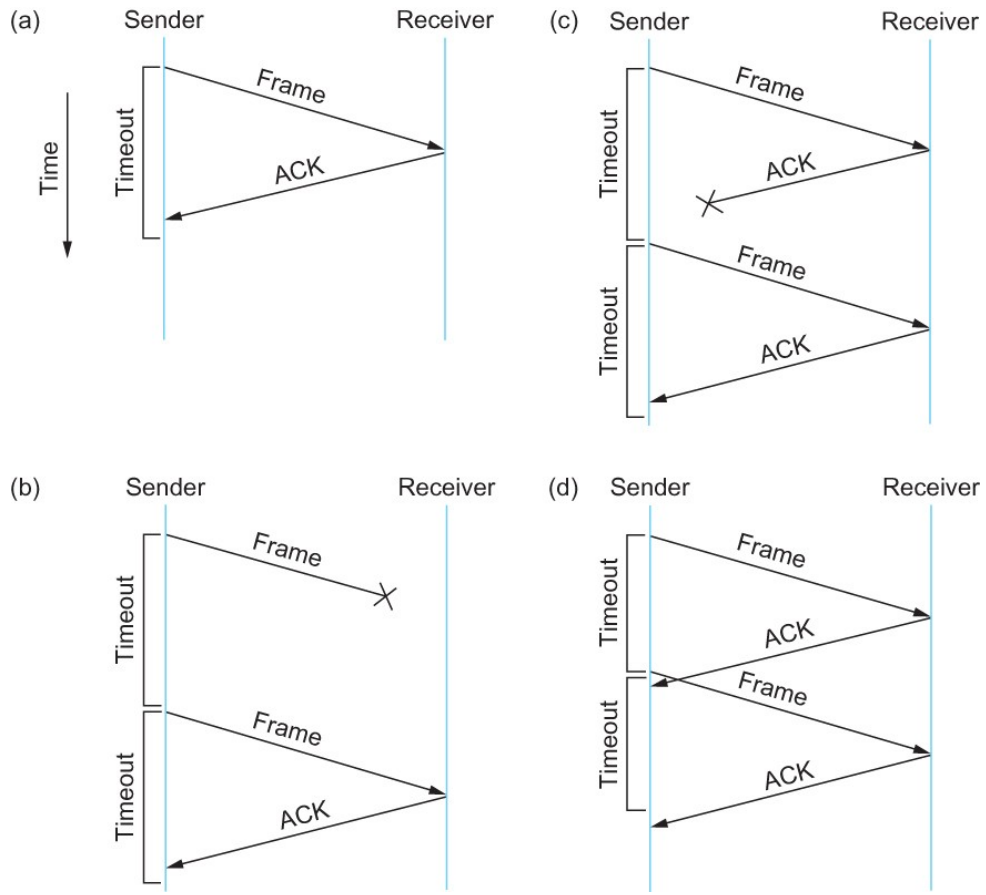
- Frames might get lost
  - Too many bits lost
  - Clock did not sync properly
  - Error detected but the report got lost
- Can we build links that does not have errors?
  - Not possible
- How about all those error correction stuff we learned?
  - Can we add them to frames?
  - We could, but think of the overhead
  - What happens when the entire frame is lost?

Frames – bag of bits



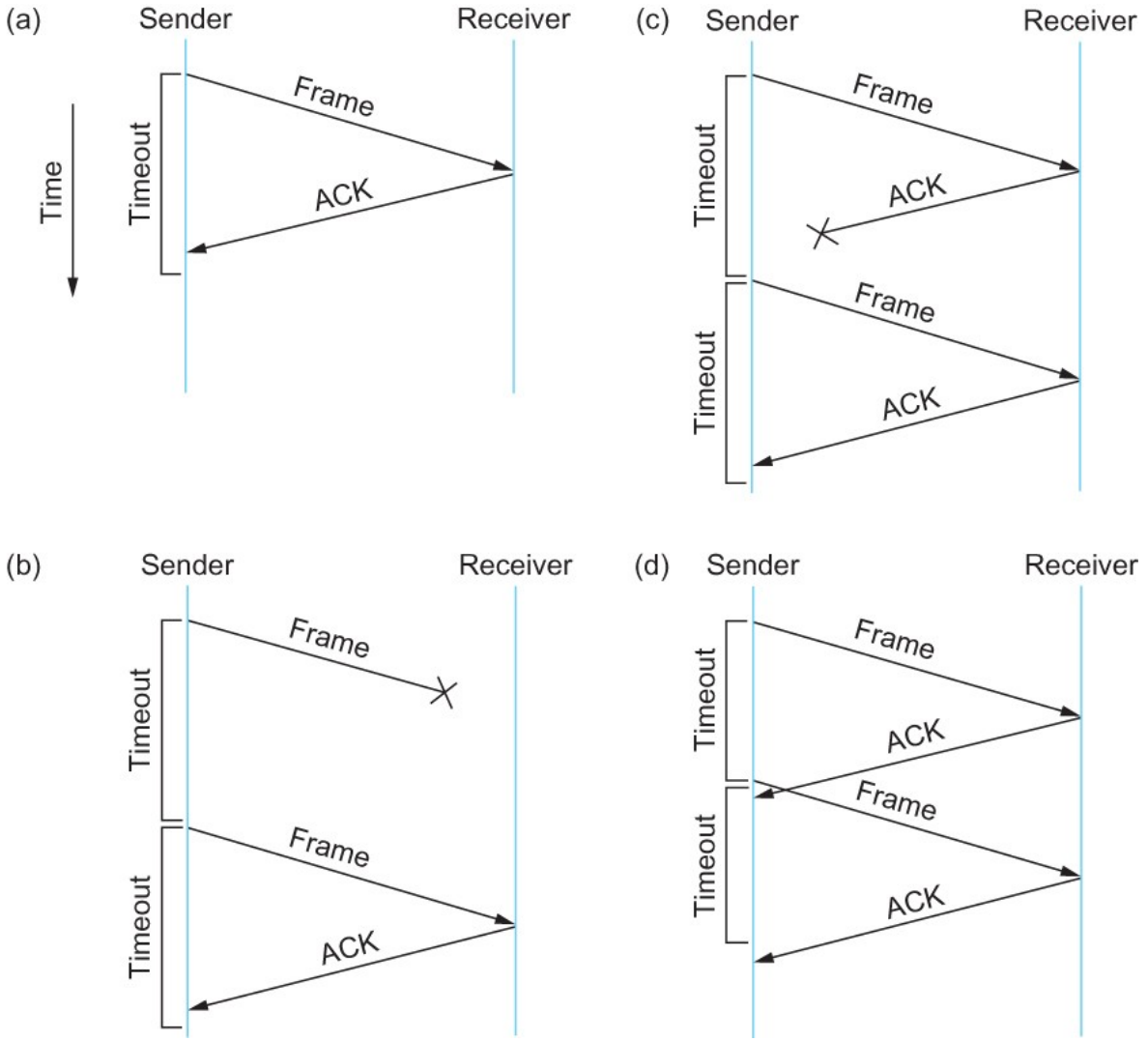
- Sending side – encapsulation, add error check bits, flow control
- Receiving side – extract frames, check for error, flow control

Stop and Wait



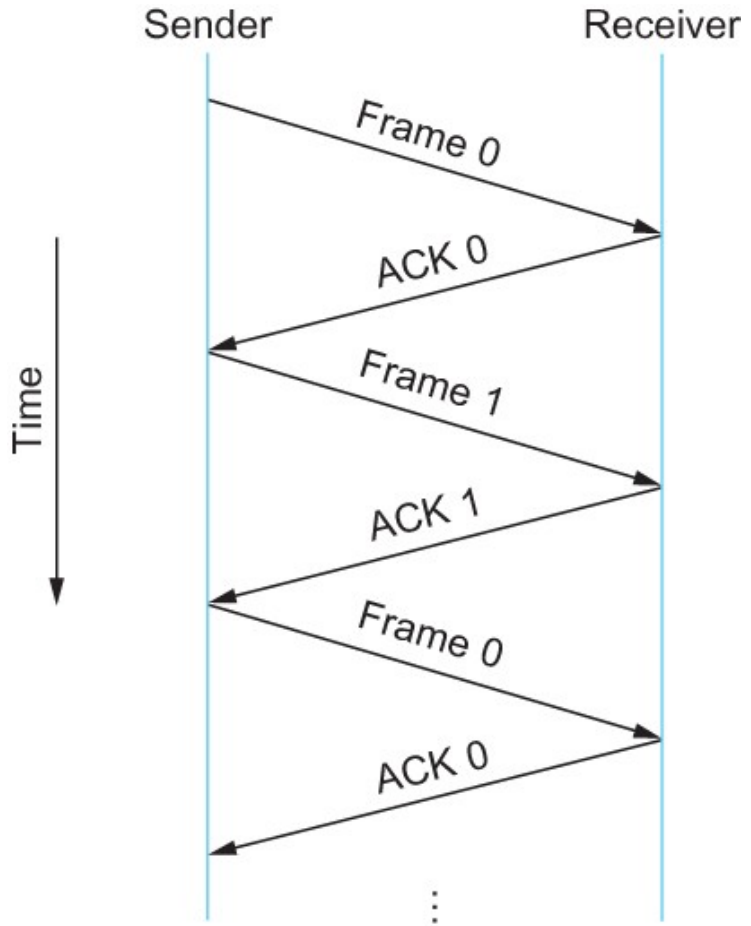
- Sender sends a frame, sets a timeout (e.g., 1 sec)
- Receiver receives the frame, sends an ACK
- Sender
  - sends the next frame on ACK
  - retransmits the same frame if timeout happens
- **Spot the bugs in the protocol**

Stop and Wait – Bugs (C and D)

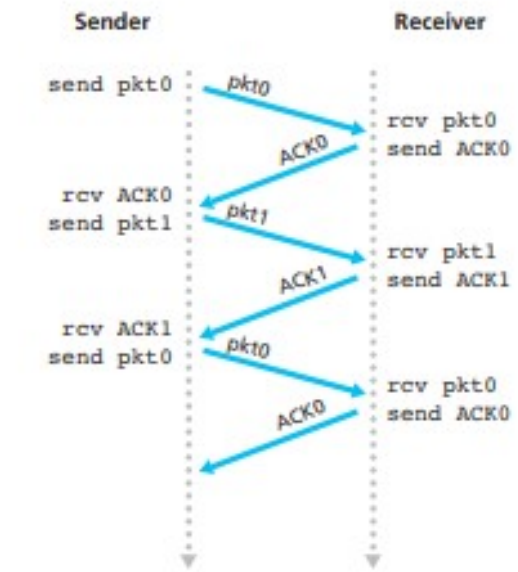


Stop and Wait – How to fix the bug?

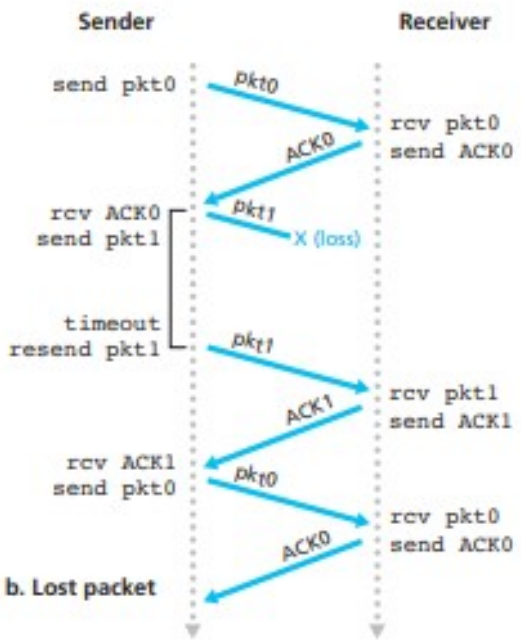
Hint: Uniquely identify each packet



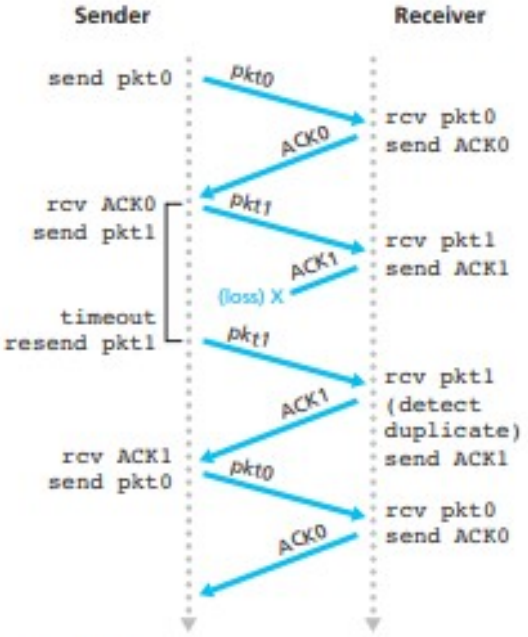
Stop and Wait v2



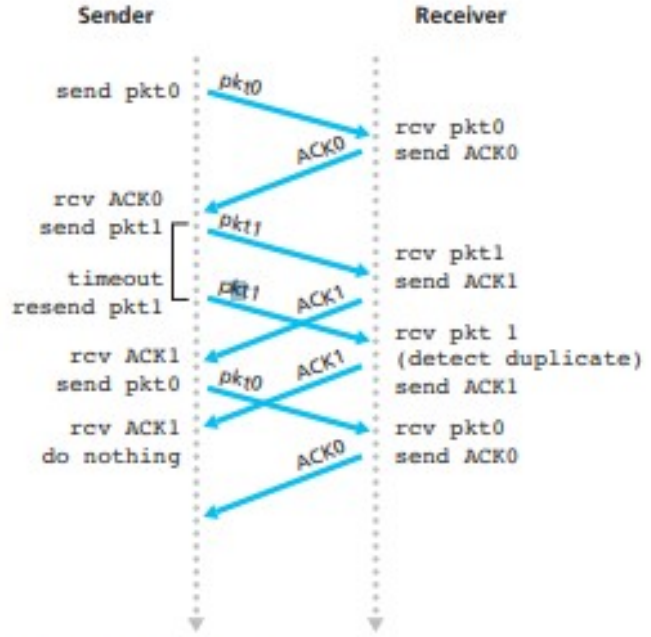
a. Operation with no loss



b. Lost packet



c. Lost ACK

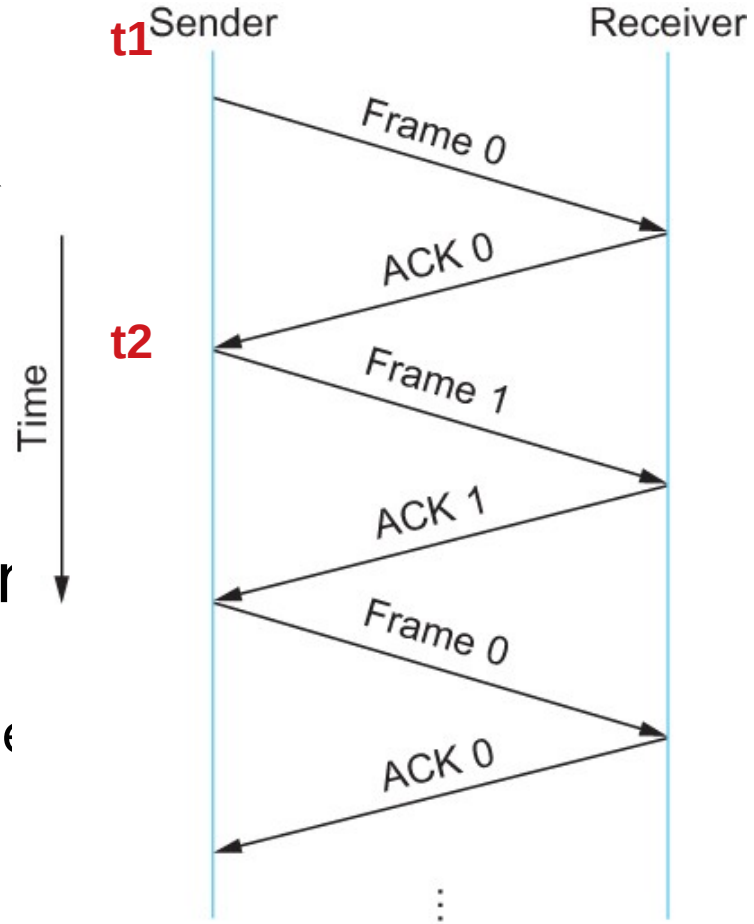


d. Premature timeout

Stop and Wait - V2 Problems

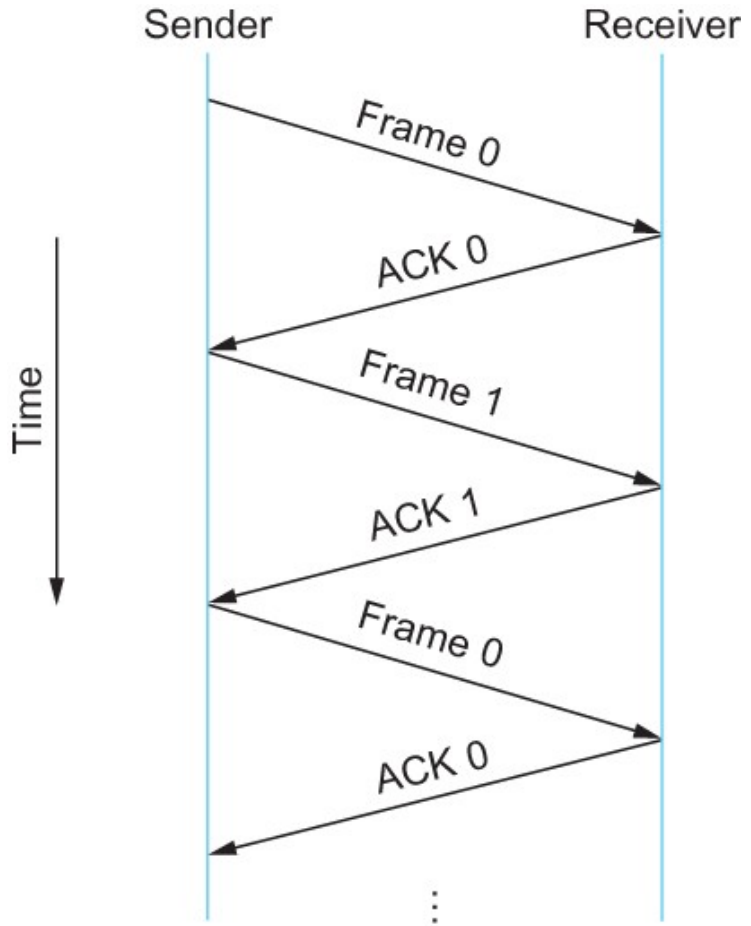
- Sender sets a timeout to wait for an ACK
  - Too small – retransmissions
  - Too large – long wait if frames are lost

- Solution:
  - Keep a running average of Round Trip Tir
    - EstimatedRTT = (1 – α) • EstimatedRTT + α • Sample
    - Timeout = 2*EstimatedRTT
    - Value of α = 0.125
    - Where does α come from? RFC 6928 (for now)



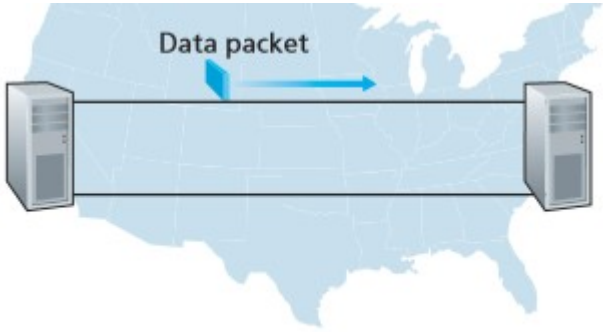
Stop and Wait – How to fix the bug?

Hint: Uniquely identify each packet



Stop and Wait – How does it perform?

- Bandwidth (R)= 1Gbps
- Packet size (L) = 1000 bytes
- RTT = 30ms
- T_{trans} = L/R = 8000bits/10^9bits/sec = 8microsecond
- T_{prop} = 15ms
- Total Delay = 15.008 ms

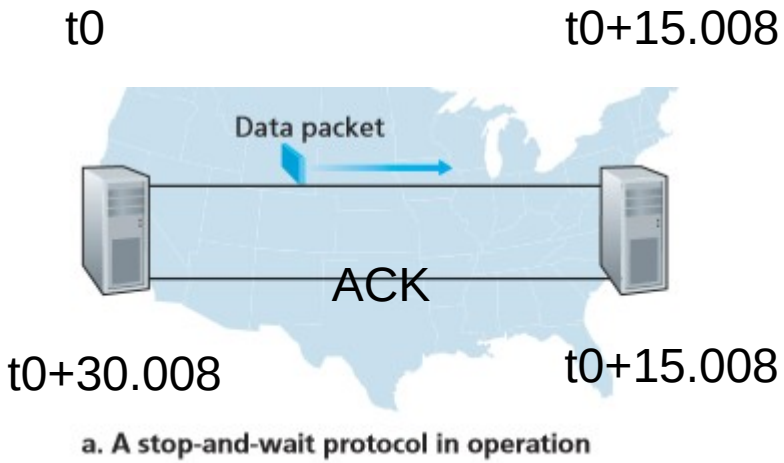


a. A stop-and-wait protocol in operation

Kurose/Ross

Stop and Wait – How does it perform?

- Sender transmits for only 0.008 ms in 30.008ms
- Utilization = 0.008/30.008 = 0.00027
- One bit at a time
- Worse when loss happens!

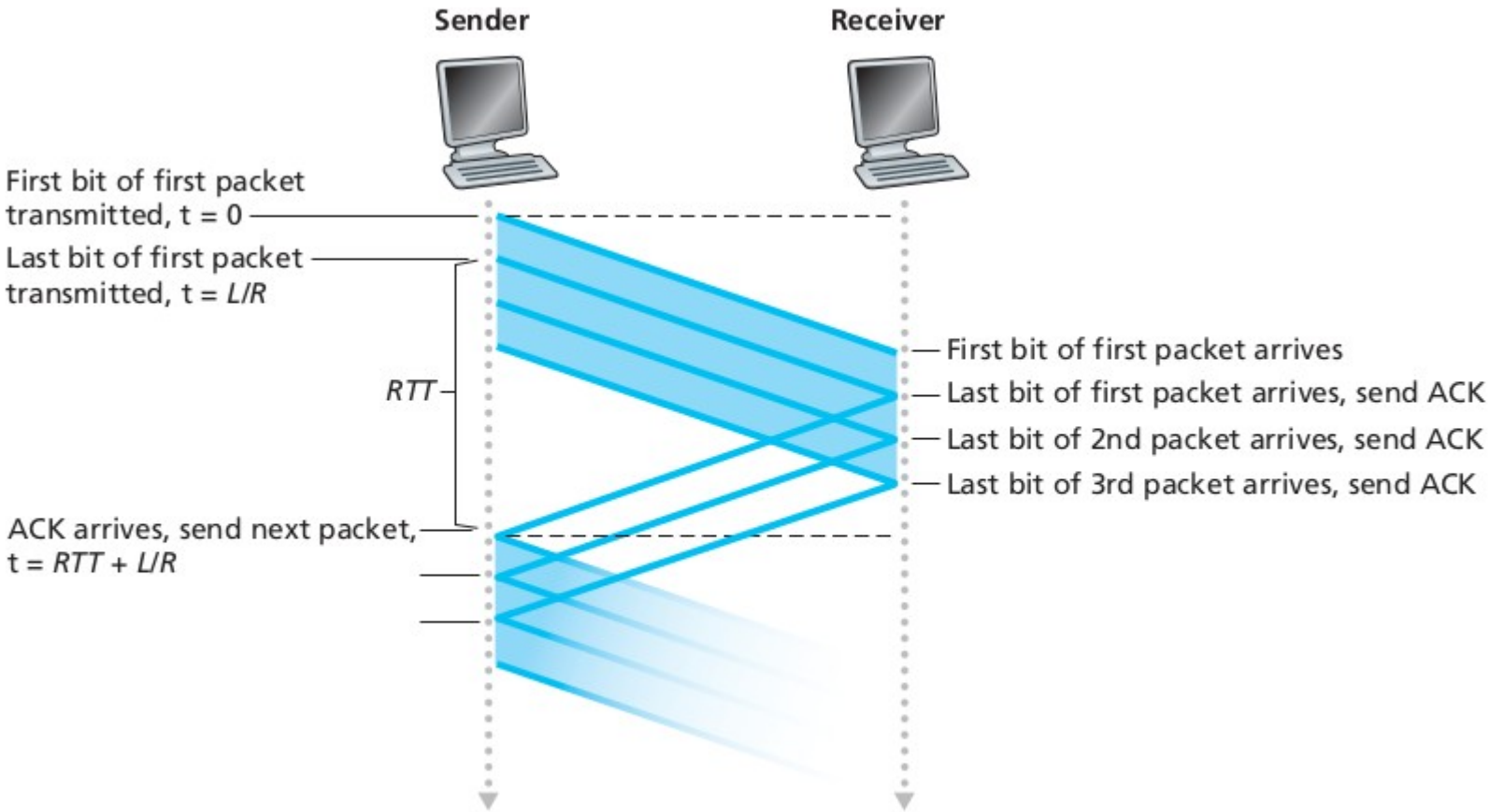


a. A stop-and-wait protocol in operation

Kurose/Ross

Sliding window to the rescue!

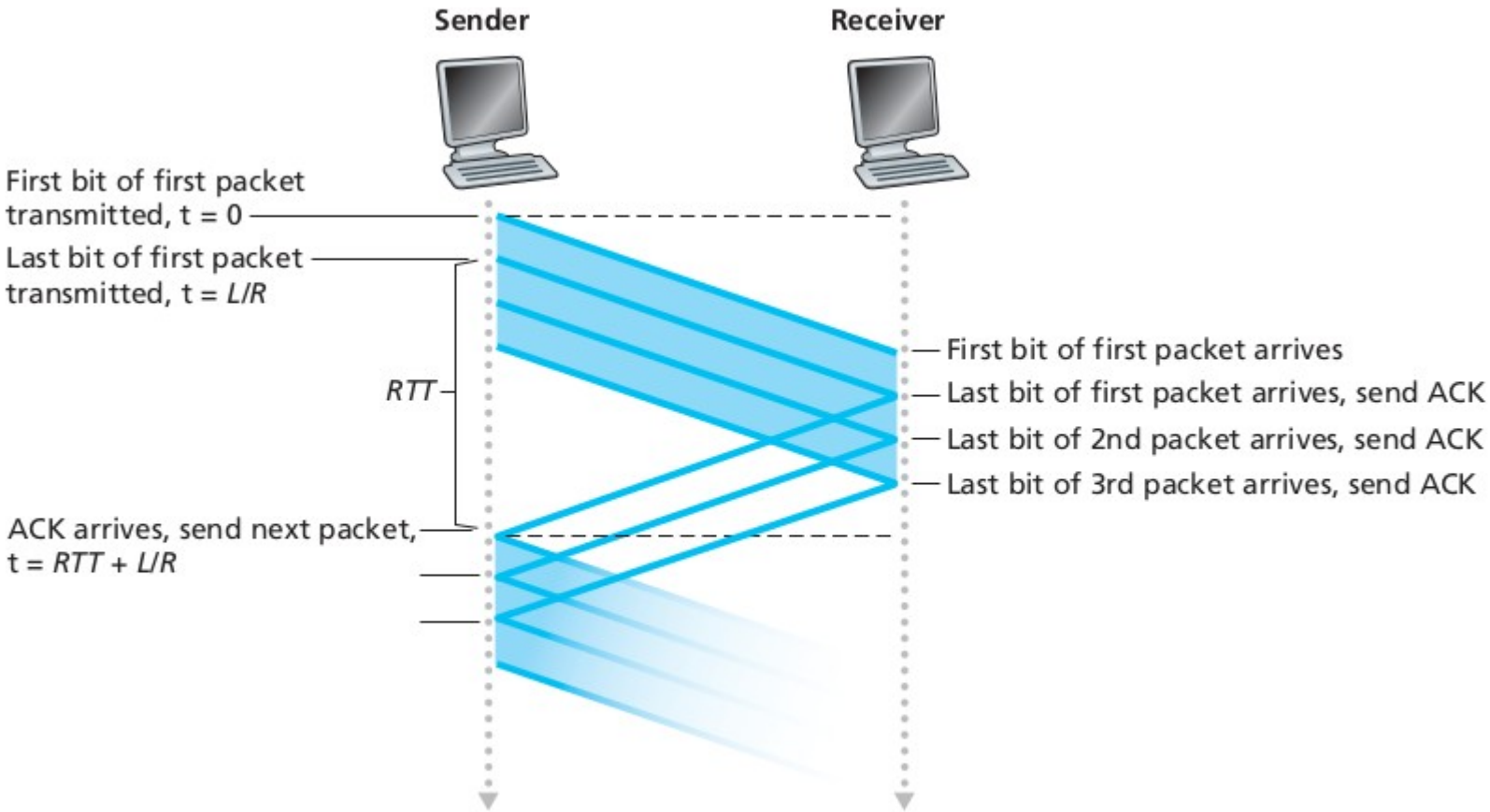
Utilization = 0.008*3/30.008 = 0.00079 (3 times increase)



b. Pipelined operation

Sliding window to the rescue!

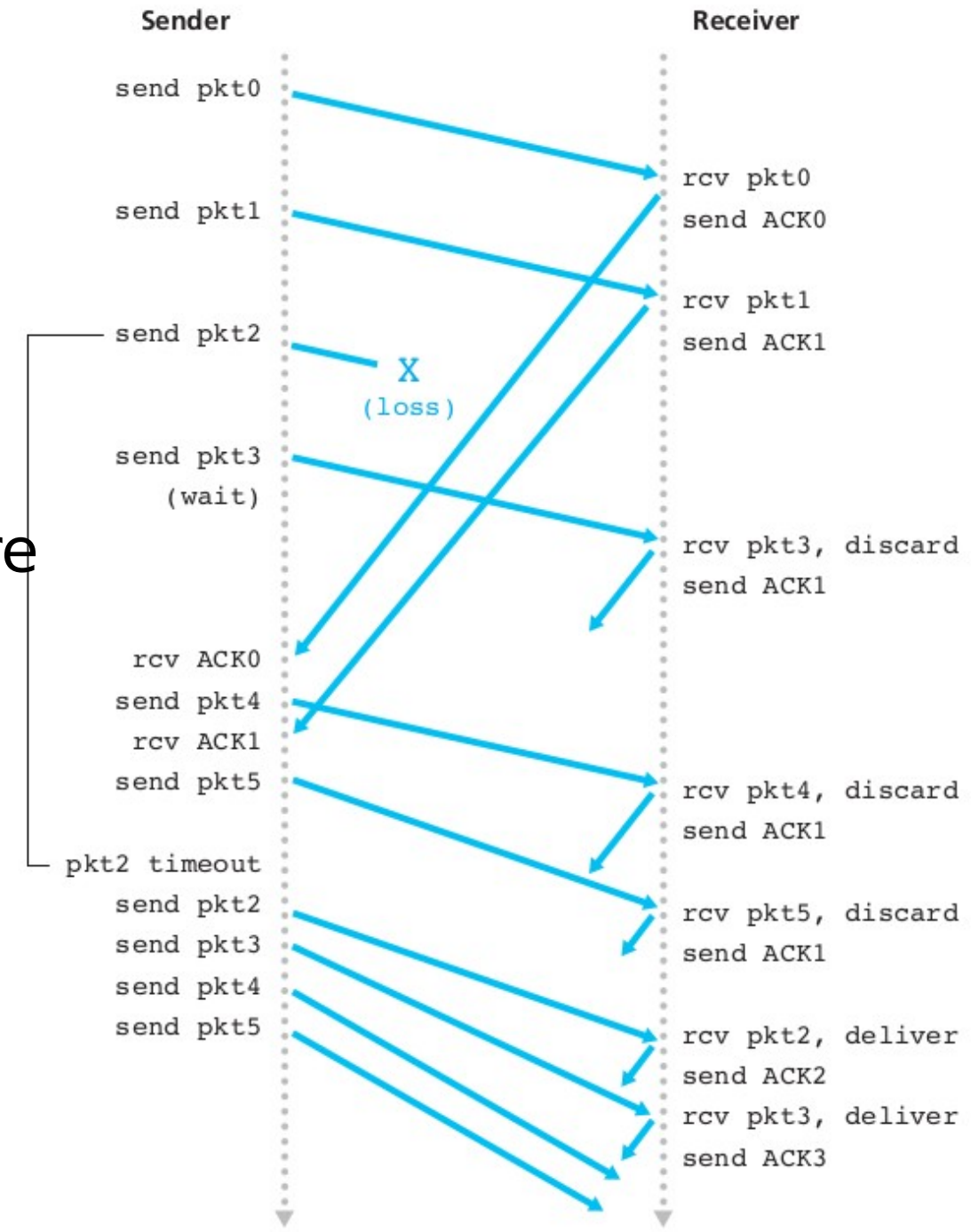
Utilization = 0.008*3/30.008 = 0.00079 (3 times increase)



b. Pipelined operation

Go-Back-N

- See the problem?
- Can not move forward until all previous packets are acknowledged



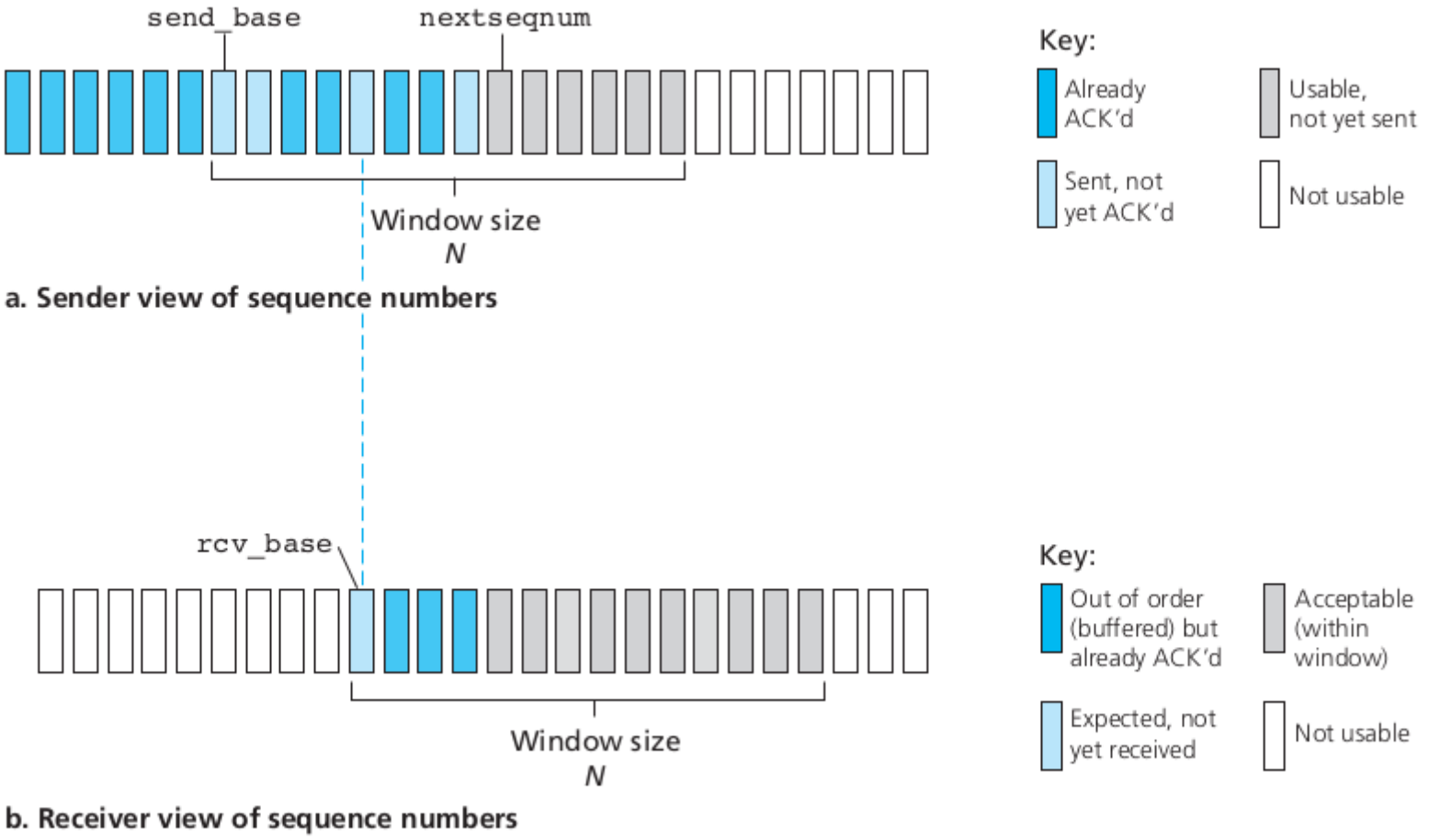
Selective Repeat

http://www.exa.unicen.edu.ar/catedras/comdat1/material/Filminas3_Practico3.swf

- Receiver:
  - Individually acks all packets
  - Buffers packets as necessary
  - Buffer packets until lost packets are received

- Sender:
  - Resend packets ***(only)*** for which ACK not received
  - Timer for each unACKed packet
  - Can send only n packets

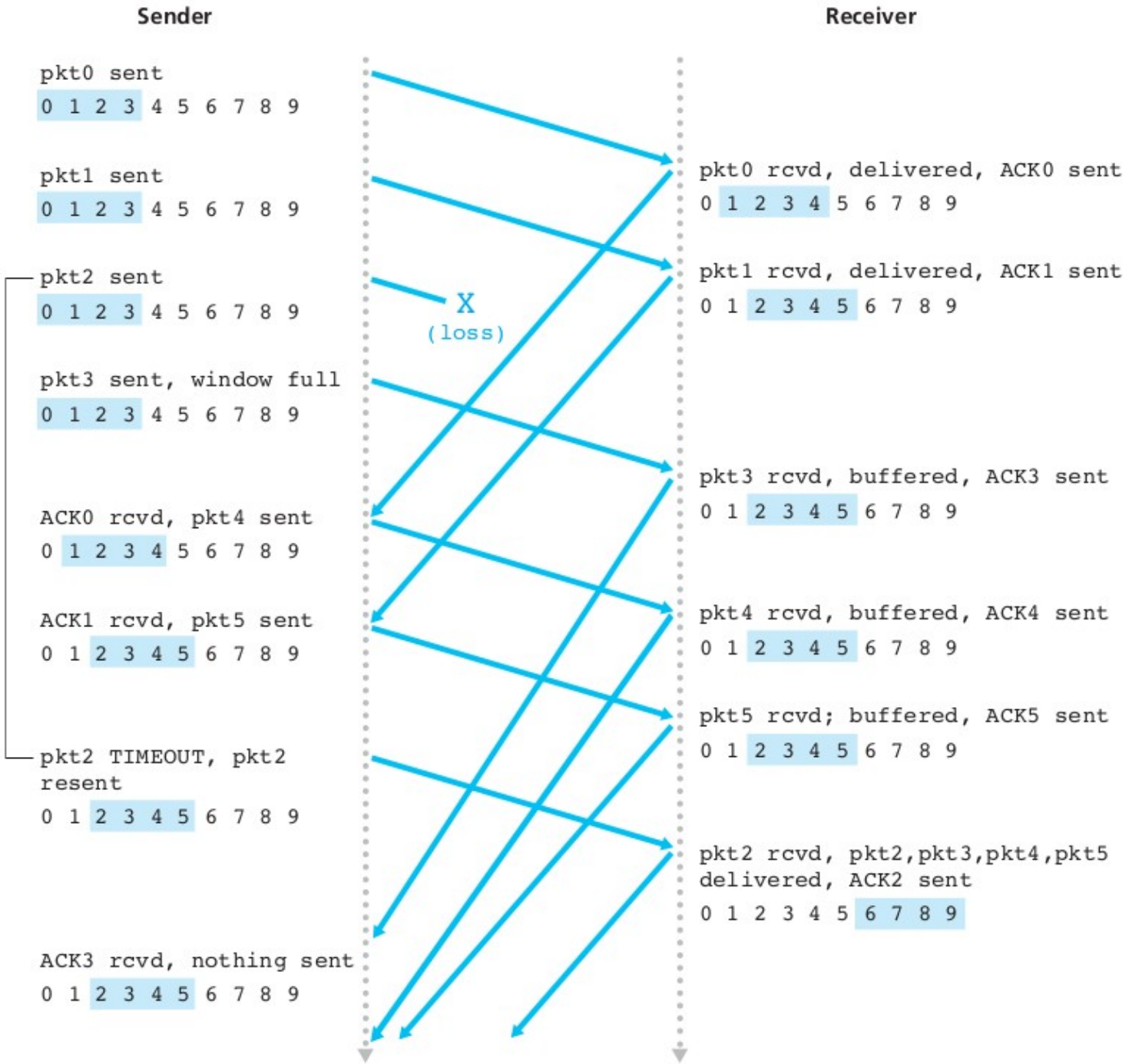
Sliding window - Selective Repeat



Sliding window – Selective Repeat - LOSS

- **Sender**:
  - Data received, if next to-be-sent-packet's seq # within window, send. Else, buffer or return to application.
  - Timeout: Each packet has its own timer. resend the packet
  - ACK received: Mark received, Advance window to next unacked seq # if ack for send_base

- **Receiver**, packet (n)
  - Sequence between recev_base, recv_base + N - 1, send ack (n)
  - Out of order: buffer
  - In-order or closes gap – deliver to application
  - Packet within <recv_base-N, recv_base -1>, ACK(n)
  - Otherwise: Ignore


Sender
Receiver
pkt0 sent
0 1 2 3 4 5 6 7 8 9
pkt1 sent
0 1 2 3 4 5 6 7 8 9
pkt2 sent
0 1 2 3 4 5 6 7 8 9
X
(loss)
pkt3 sent, window full
0 1 2 3 4 5 6 7 8 9
ACK0 rcvd, pkt4 sent
0 1 2 3 4 5 6 7 8 9
ACK1 rcvd, pkt5 sent
0 1 2 3 4 5 6 7 8 9
pkt2 TIMEOUT, pkt2
resent
0 1 2 3 4 5 6 7 8 9
ACK3 rcvd, nothing sent
0 1 2 3 4 5 6 7 8 9
pkt0 rcvd, delivered, ACK0 sent
0 1 2 3 4 5 6 7 8 9
pkt1 rcvd, delivered, ACK1 sent
0 1 2 3 4 5 6 7 8 9
pkt3 rcvd, buffered, ACK3 sent
0 1 2 3 4 5 6 7 8 9
pkt4 rcvd, buffered, ACK4 sent
0 1 2 3 4 5 6 7 8 9
pkt5 rcvd; buffered, ACK5 sent
0 1 2 3 4 5 6 7 8 9
pkt2 rcvd, pkt2,pkt3,pkt4,pkt5
delivered, ACK2 sent
0 1 2 3 4 5 6 7 8 9

Issues with Sliding Window Protocol

- When timeout occurs, the amount of data in transit decreases
  - Since the sender is unable to advance its window
- When the packet loss occurs, this scheme is no longer keeping the pipe full
  - The longer it takes to notice that a packet loss has occurred, the more severe the problem becomes
- How to improve this
  - Negative Acknowledgement (NAK)
  - Additional Acknowledgement
  - Selective Acknowledgement (SAK)

Next Steps

- Read Through - Chapter 2.5.2
- Ethernet and WiFi
- Project 1 due on the 20th
- Project 2 and homework 2 will post on the 20th