SUNWAY UNIVERSITY

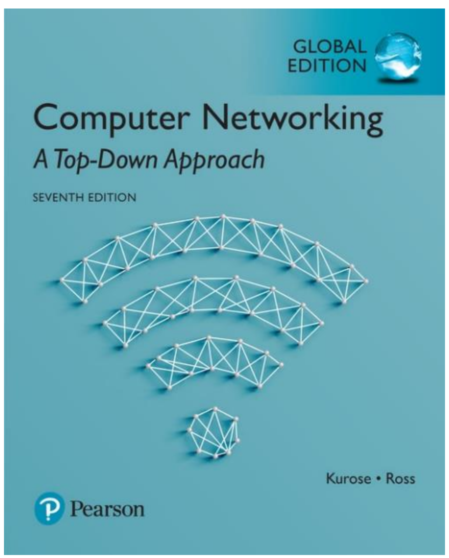
Slides Production: Prof. Yau Kok Lim

Dr. Saad Aslam

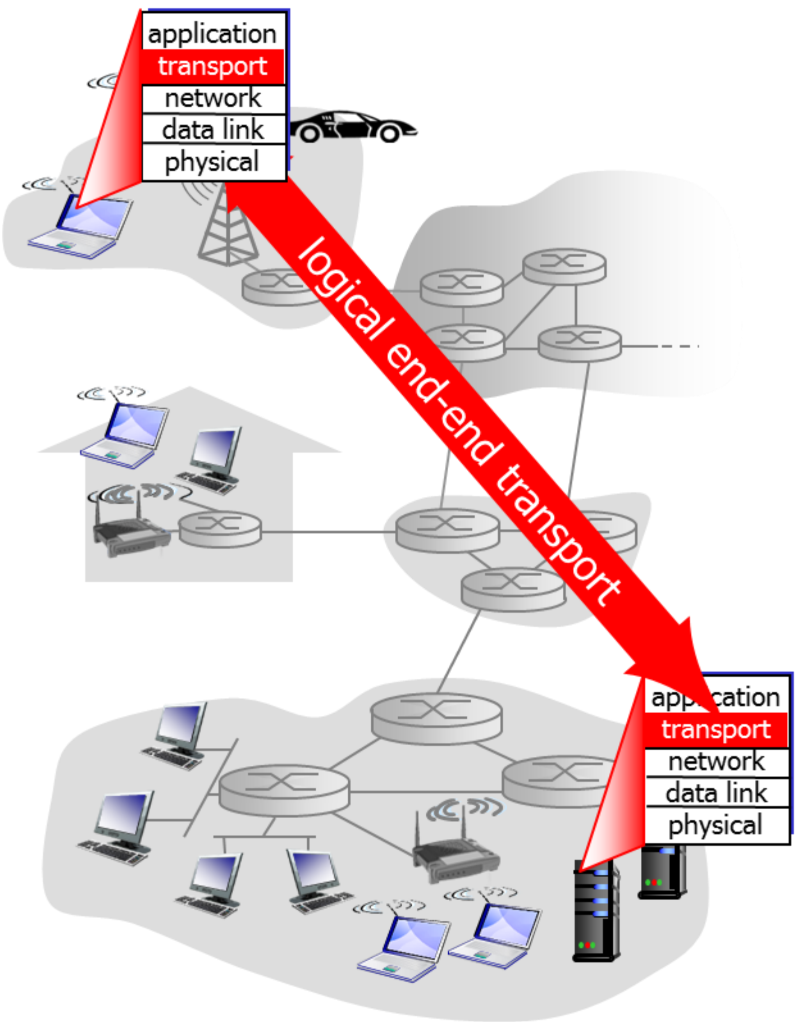
Office: Room AE-327, 3rd Floor, NUB

Contact: +603 74918622
(Ext: 7143)

Email: saada@sunway.edu.my



Computer Networking: A Top-down Approach, 7th edition.
Jim Kurose, Keith Ross
Pearson



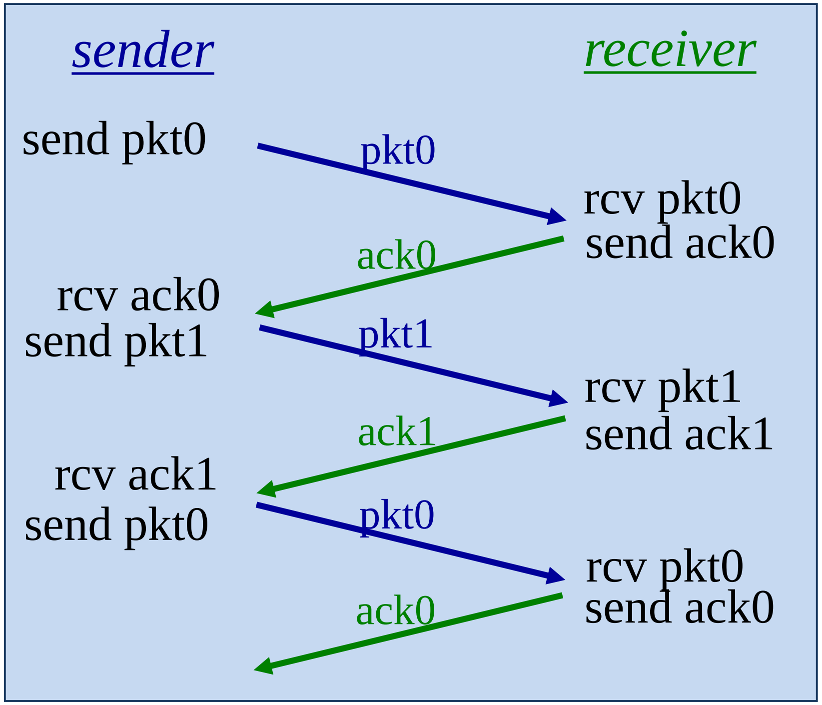
- **Transport Layer**
  - Provide logical communication between processes running on different hosts
    - Sender host: Break a message into segments
    - Destination host: Reassemble segments into the message, and pass to application layer
  - Rely on network layer services
    - *network layer*
      - Provide logical communication between hosts
  - Two type of protocol
    - **Transmission Control Protocol (TCP)**
    - **User Datagram Protocol (UDP)**

SUNWAY UNIVERSITY

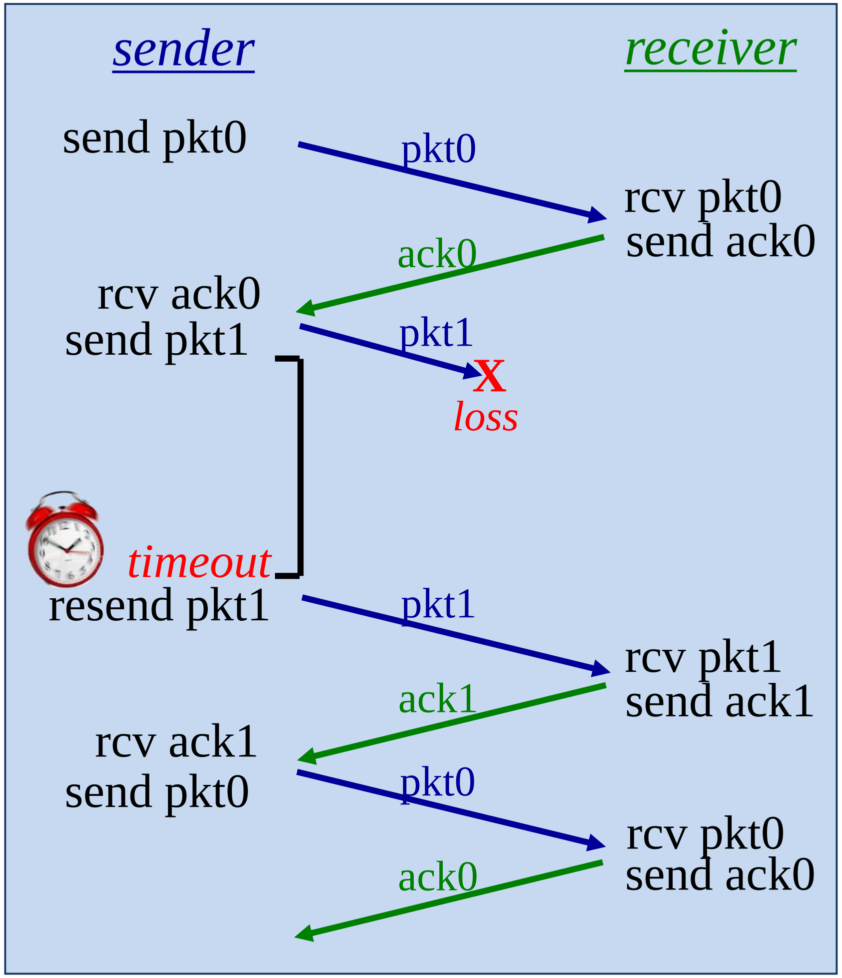
Comparison between TCP and UDP

<table>
<tr><th>TCP</th><th>UDP</th></tr>
<tr><td>Reliable
Destination host receive all data in-order
Use Go-Back-N</td><td>Unreliable
Destination host may not receive some lost data and out-of-order</td></tr>
<tr><td>Pipelined
Congestion control
Resend segment until successful transmission
Slower: Retransmission incurs delay
Control a source host’s sending rate so that each connection traversing a congested link get an equal share of link bandwidth
Flow control
Control a source host’s sending rate so that it does not overflow receive host’s buffer</td><td>No pipelined
No congestion control
Do not resend segment until successful transmission
Faster
Do not control a source host’s sending rate</td></tr>
<tr><td>Connection-oriented
Establish connection, then start transmission
Establishing connection require handshaking (exchange of control messages to initialize sender and receiver state)
Each connection is full duplex (bi-directional data flow)
Slower: Establishing connection incurs delay
Point-to-point
One sender, one receiver</td><td>Connectionless
Do not establish connection
Faster</td></tr>
<tr><td>Stateful
Host maintain buffer utilization, congestion control parameter, acknowledgment number</td><td>Stateless
Host do not maintain buffer utilization, congestion control parameter, acknowledgment number</td></tr>
<tr><td>Packet header
Larger</td><td>Packet header
Smaller</td></tr>
<tr><td>Example of usage
Email</td><td>Example of usage
Audio and video traffic</td></tr>
</table>

- Consider FOUR scenarios
  - No packet loss
  - Packet loss
  - ACK loss
  - Timeout / Delayed ACK

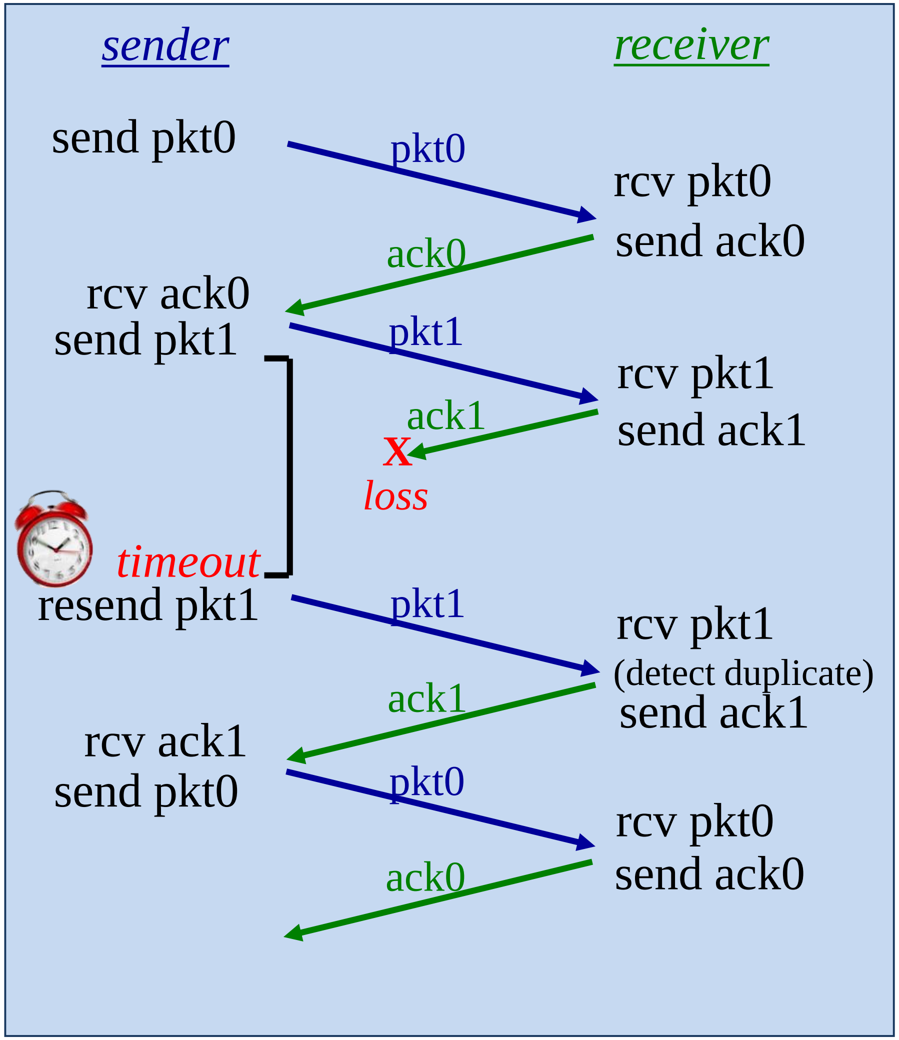


(a) No packet loss

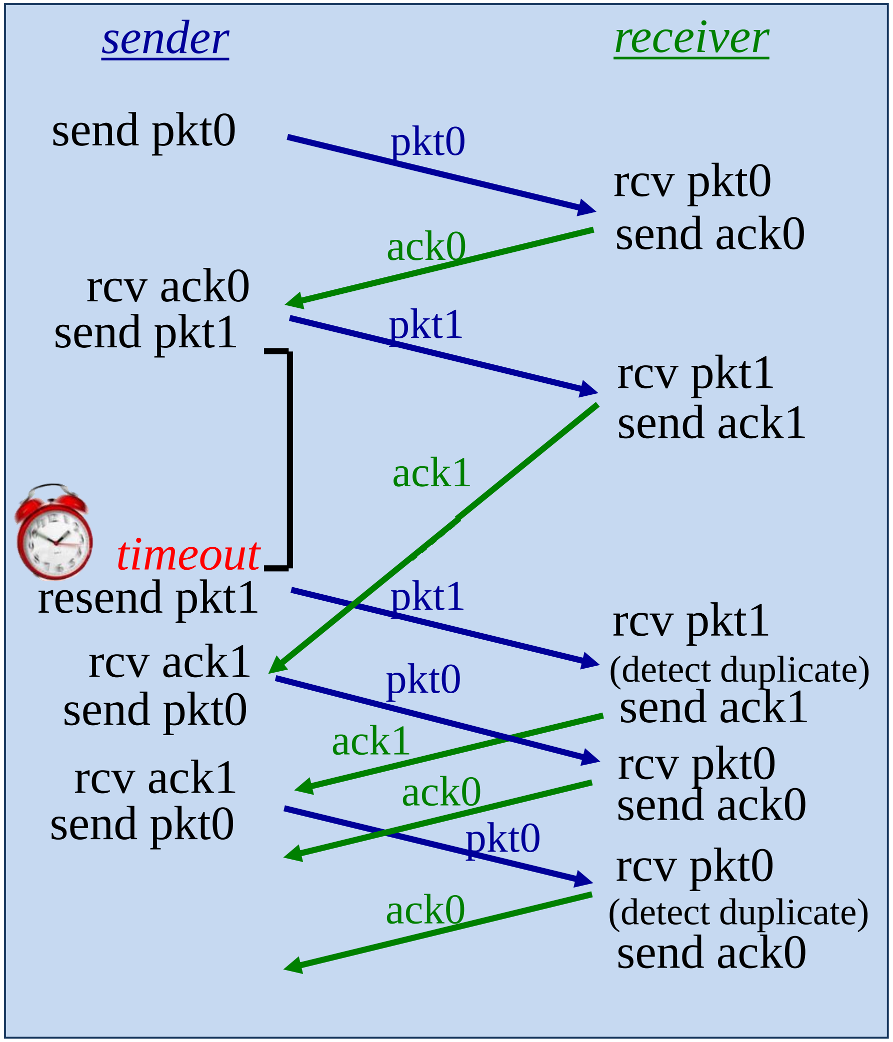


(b) Packet loss

Note: Timer is restarted when sender receive ack

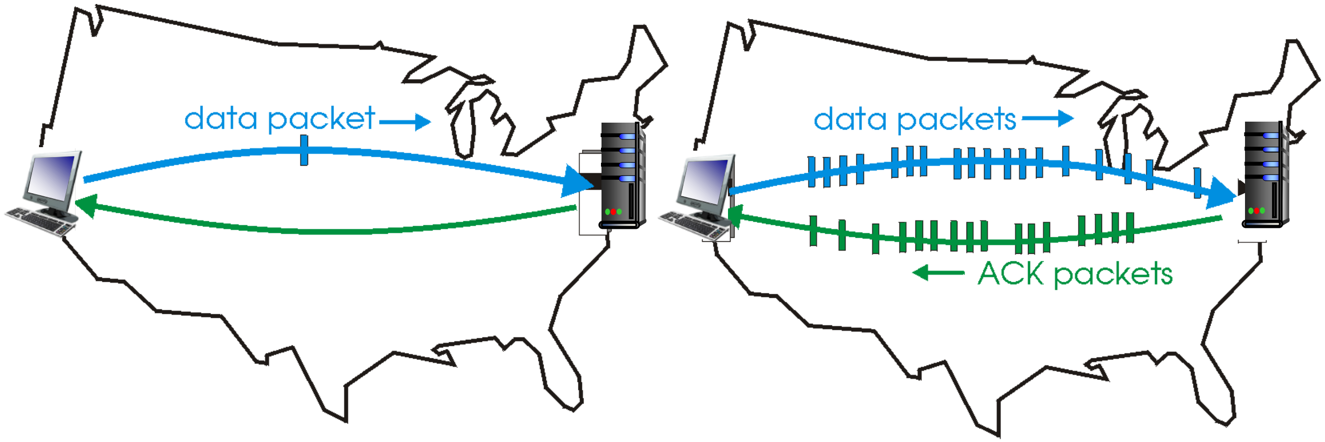


(c) ACK loss



(d) Timeout / Delayed ACK

3.4 Reliable Data Transfer Protocol:
Comparison between stop-and-wait and pipelined protocols



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

- **Stop-and-wait protocol**
  - Sender send a single packet, receive acknowledgment; then only send the next packet
- **Pipelined protocol**
  - Sender send multiple unacknowledged packets
    - Must increase the range of sequence number
    - Must have sufficient buffer size at sender and receiver
  - Two types
    - **Go-Back-N (GBN)**
    - **Selective Repeat (SR)**

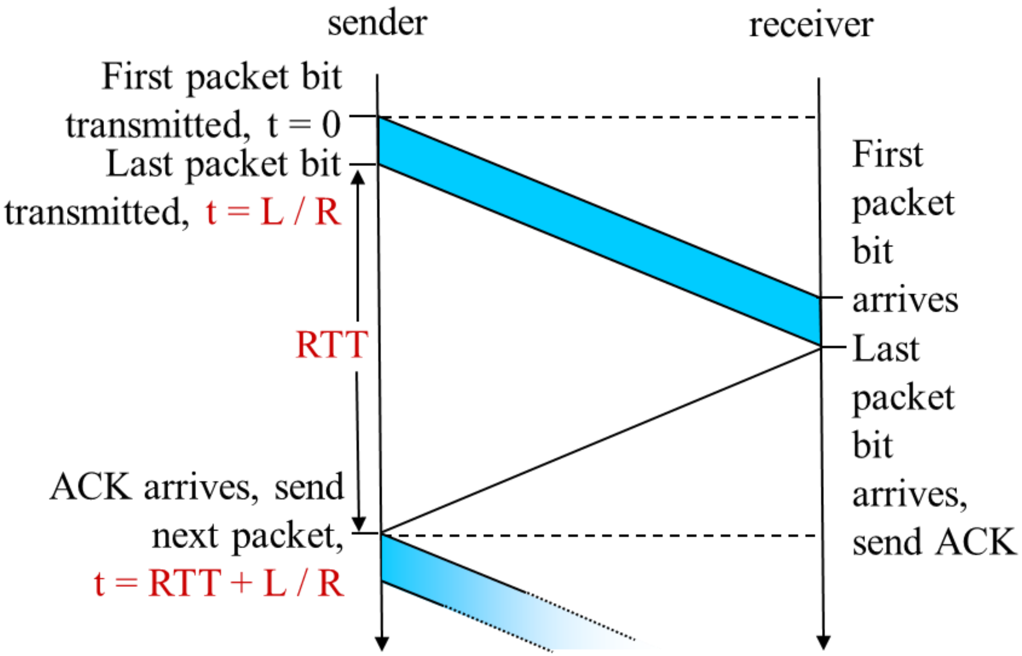
- **Stop-and-wait protocol**
  - Low utilization
  - Example:
    - Packet size, L: 8000 bit
    - Transmission rate, R: 1 Gbps
    - Propagation delay: 15 ms
    - Round trip time, RTT: 30 ms
  - What is the utilization?
    - Transmission delay
      - Time to transmit packet into the 1 Gbps link

$$d_{trans} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bits/sec} = 8\mu s$$

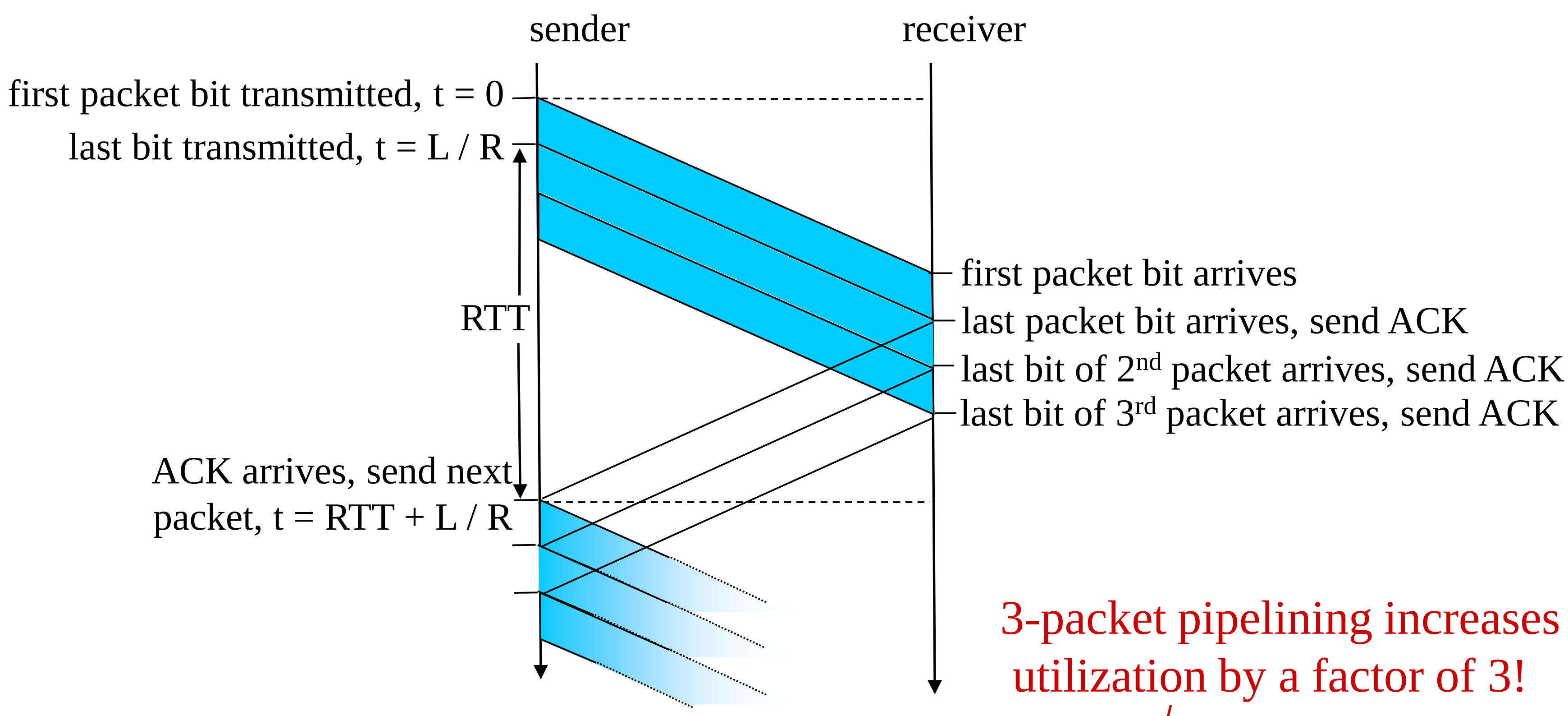
- Utilization

$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{0.008}{30.008} = 0.00027$$

- So, the effective throughput is 270 kbps only for a 1 Gbps link!



- **Pipelined protocol**
  - Higher utilization



3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3L/R}{RTT+L/R} = \frac{0.0024}{30.008} = 0.00081$$

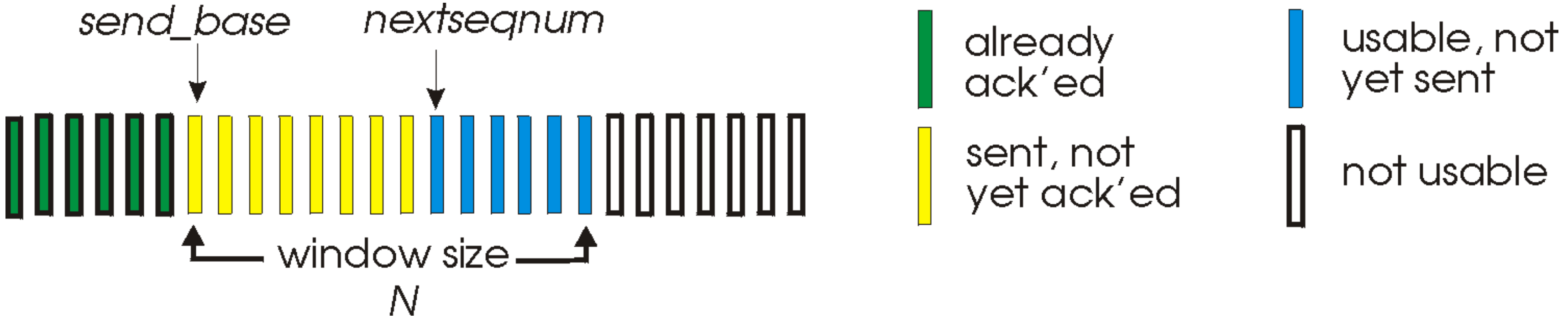
3.4 Reliable Data Transfer Protocol: Pipelined protocols

- **Go-back-N**
  - Sender
    - Can send *N* unacknowledged packets in pipeline
  - Receiver
    - Only send cumulative acknowledgment
      - This means an acknowledgment for packet *n* indicates all packets with sequence number up to and including *n* have been received correctly
    - No buffer
  - Timeout
    - Sender has a single timer for the oldest unacknowledged packet
    - When timer expire, retransmit all sent but unacknowledged packets
      - If window size is large
        - There can be many unacknowledged packets in pipeline, so a single packet loss can cause sender to retransmit many packets

- **Selective Repeat**
  - Sender
    - Can send *N* unacknowledged packets in pipeline
  - Receiver
    - Send acknowledgment for each packet
    - Has buffer
  - Timeout
    - Sender has timer for each unacknowledged packet, so it has many timers
    - When timer expire, retransmit the unacknowledged packet only
      - Address the disadvantage of Go-back-N

3.4 Go-Back-N (GBN)

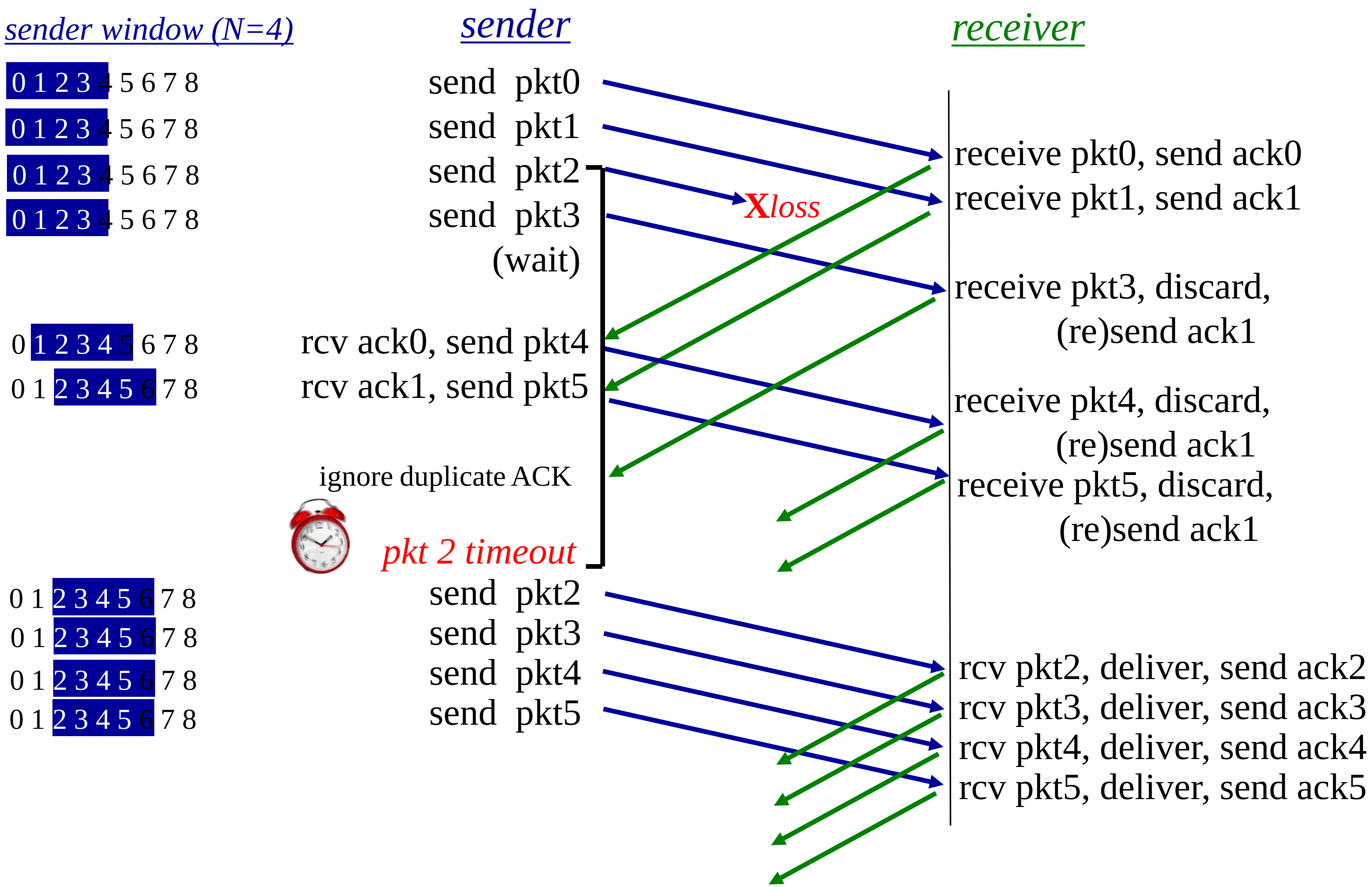
- Go-back-N
  - Maintain a window
    - Window size = *N*
      - A sender can send *N* unacknowledged packets in pipeline
    - When sender receive an acknowledgment, it slide forward window so that a sequence number change from *not usable* to *usable*, *not yet sent*
  - Each packet has sequence number
    - Sequence number



3.4 Go-Back-N (GBN)

- Go-back-N
  - Receiver
    - Always send ACK for correctly-received packet with the highest in-order sequence number
    - Generate duplicate ACKs when receive out-of-order packet
        - Discard out-of-order packet because receiver does not have buffer
    - Re-acknowledge packet with the highest in-order sequence number
      - E.g.:
        - Suppose,
          - Receiver
            - Expect to receive packet *n*
            - However, it receive packet *n*+1. This mean that packet *n* is lost
            - Discard packet *n*+1
            - Resend acknowledgement for packet *n*-1
          - Sender
            - Receive duplicated ACKs
            - Resend packet *n* and *n*+1

3.4 Go-Back-N (GBN)



sender

Data from upper layer

- If there is *usable*, *not yet sent* in window, send the packet

Timeout of packet *n*

- Resend packet *n*, restart timer of packet *n*

Receive acknowledge for packet *n*

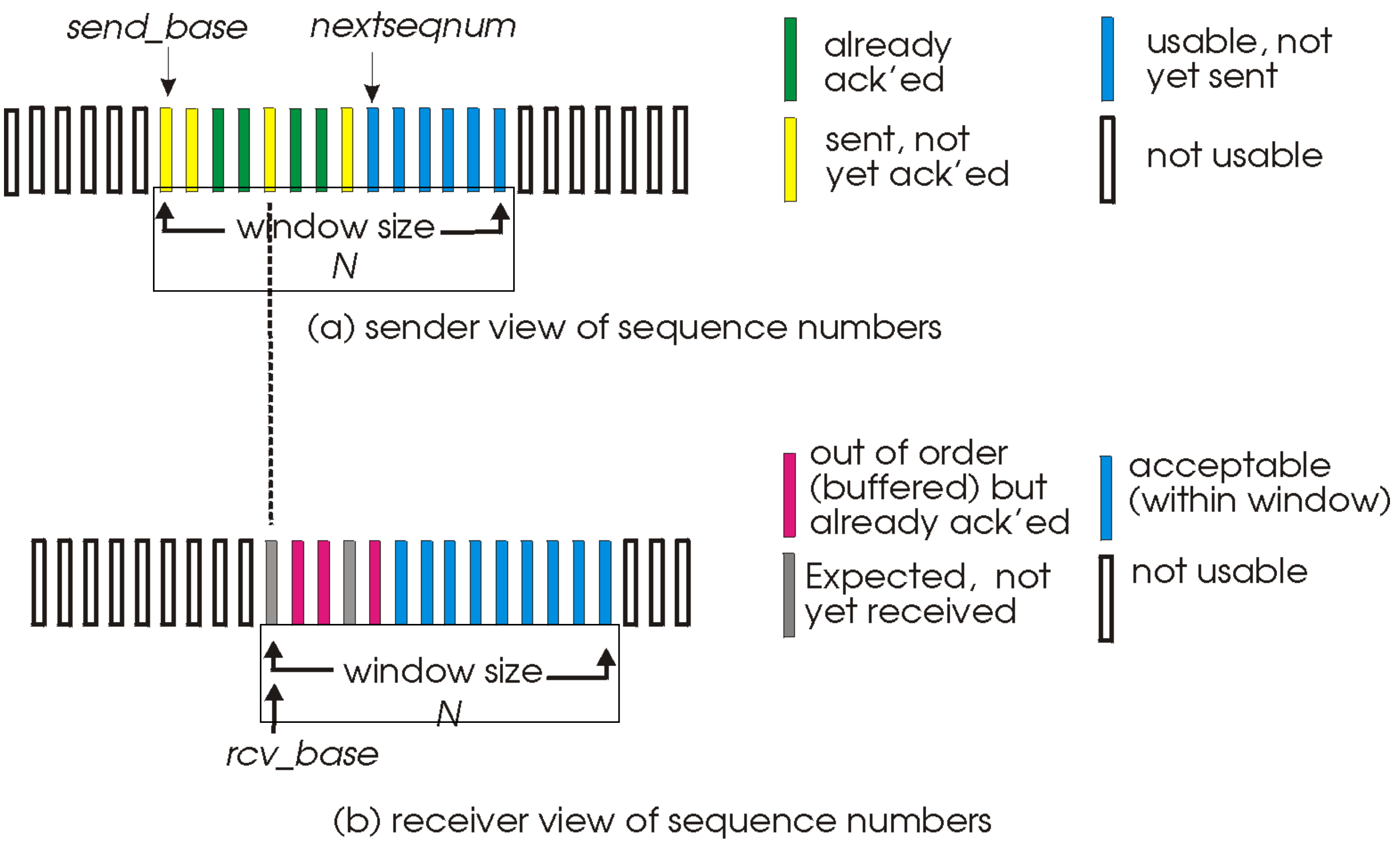
- When sender receive an acknowledgment for send_base packet, it slide forward window so that a sequence number change from not usable to usable, not yet sent

receiver

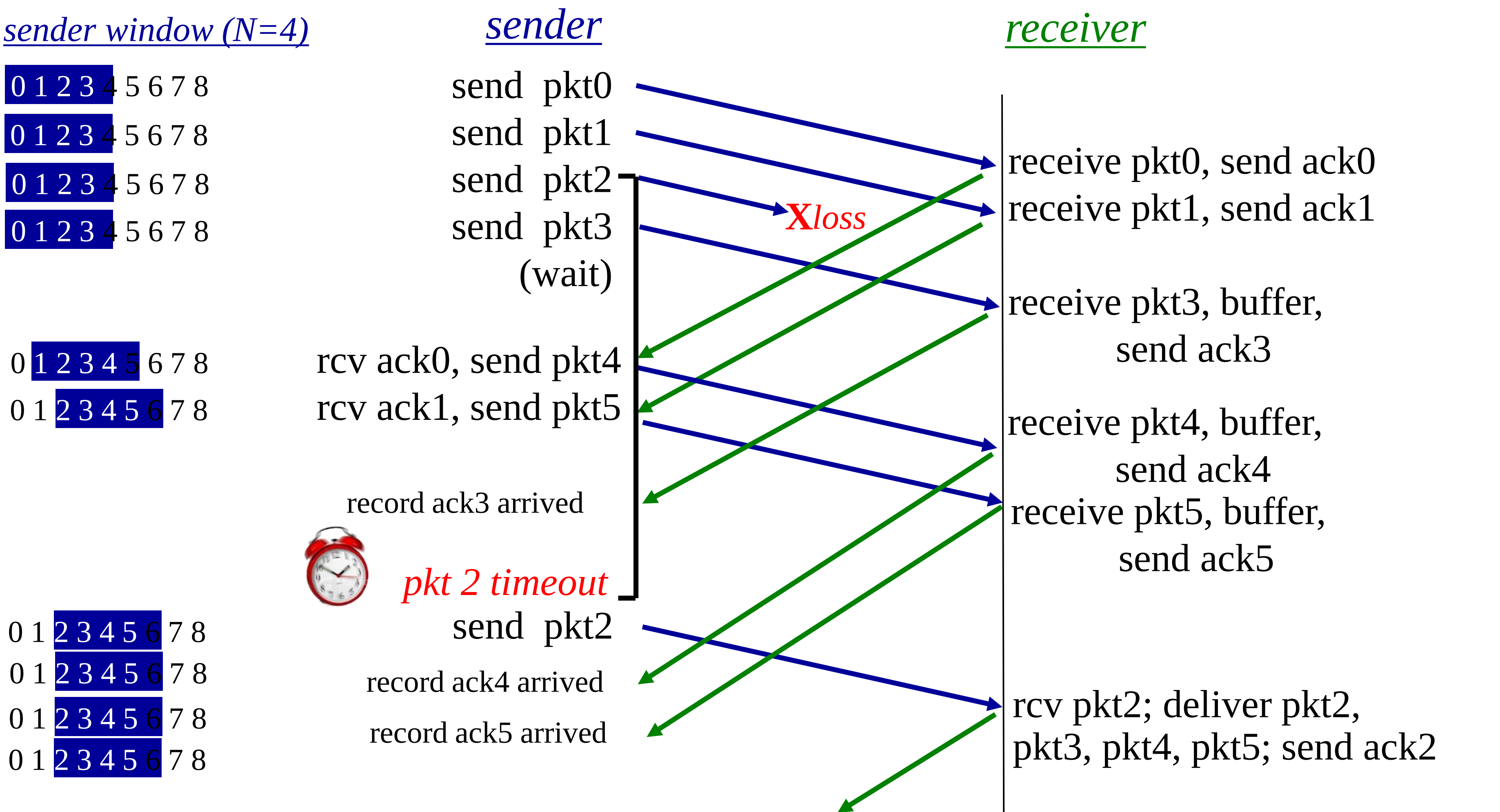
Receive packet *n*

- Send an acknowledgment for packet *n*
- If
  - Out-of-order packet
    - Buffer the packet
  - In-order packet
    - Deliver (also deliver buffered, in-order packets) to upper layer

3.4 Selective Repeat (SR)



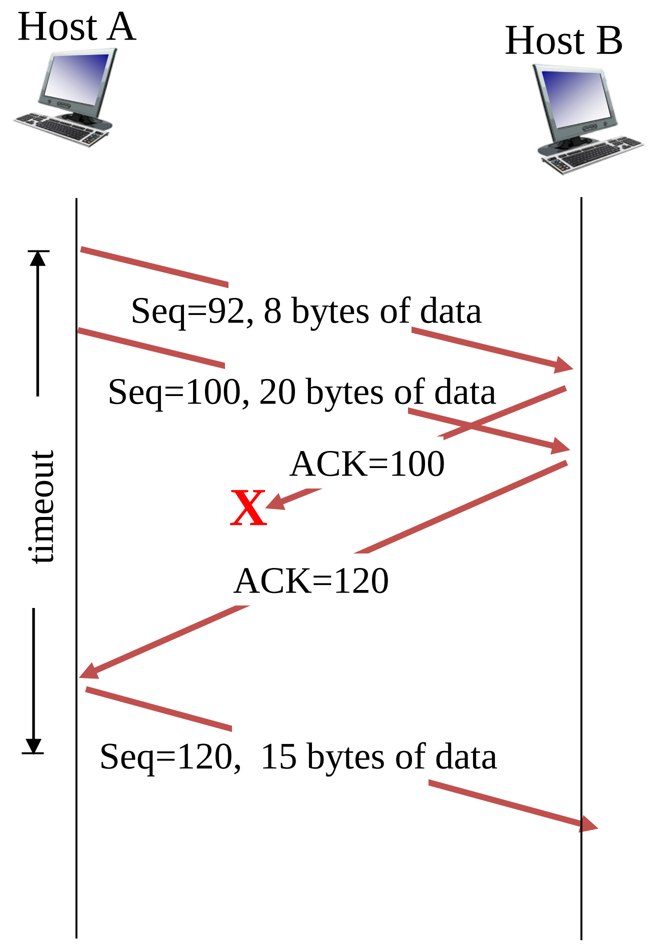
3.4 Selective Repeat (SR)



Q: what happens when ack2 arrives?

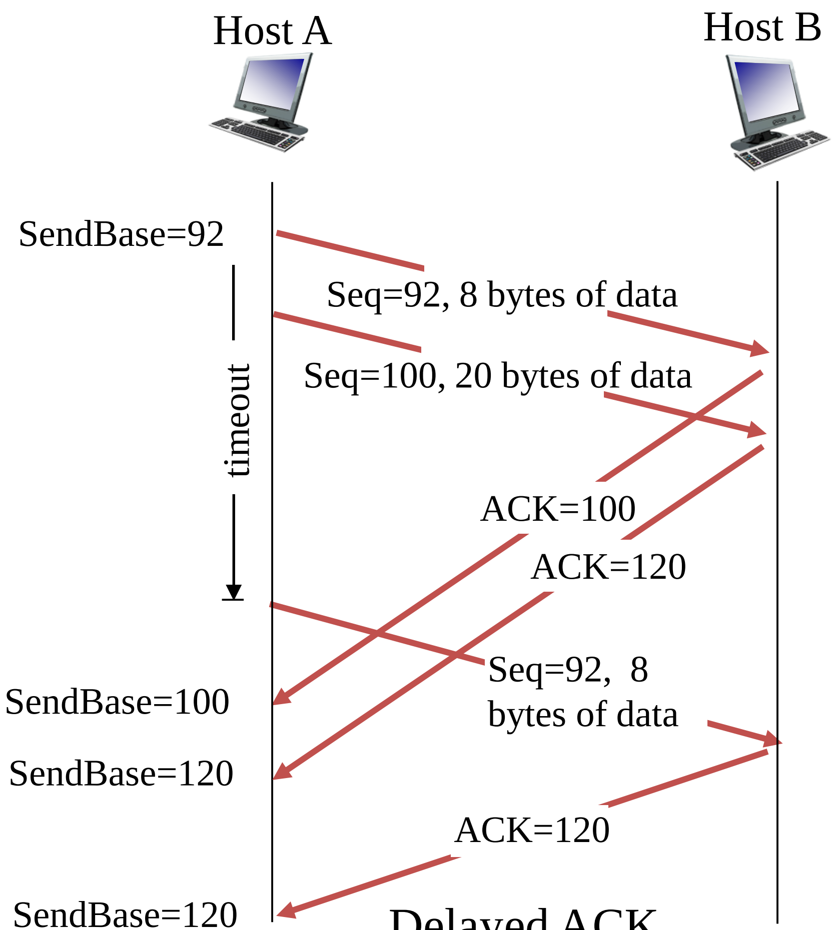
Reliable Data Transfer: Use Go-Back-N

- Use Go-Back-N
  - Receiver
    - Only send cumulative acknowledgment

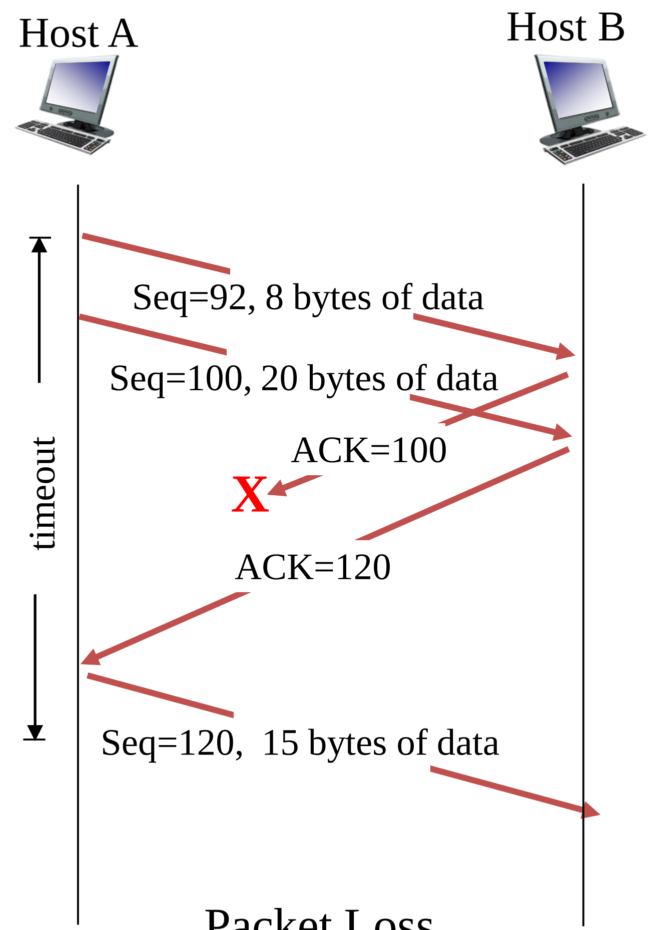


cumulative ACK

3.5 Connection-Oriented Transport: TCP
Reliable Data Transfer: TCP Retransmission Scenario



Delayed ACK



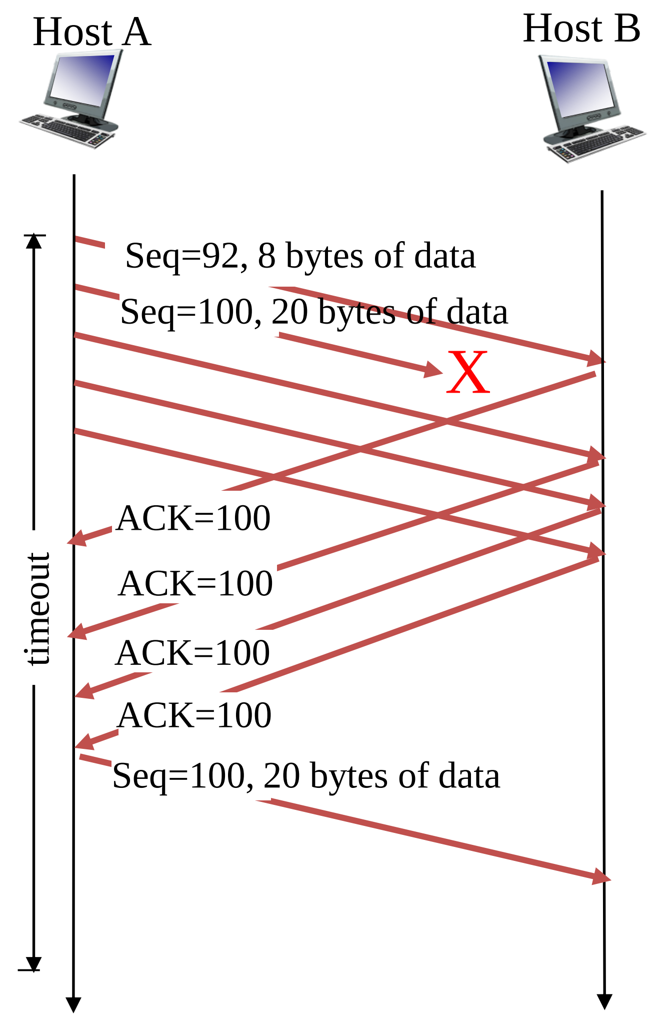
Packet Loss

Host A receive ACK = 120 before timer for the retransmission of packet Seq = 92 timeout. This indicate packet Seq = 100 has been received correctly, so packet Seq = 100 is not retransmitted

Host A receive ACK = 120. This indicate packet Seq = 92 and Seq = 100 have been received correctly. So, even though ACK = 100 is lost, packet Seq = 92 is not retransmitted

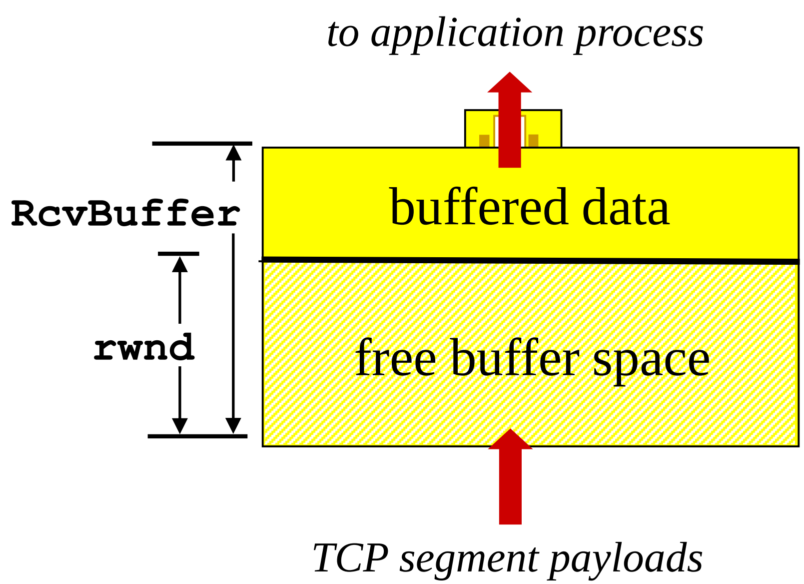
Reliable Data Transfer: Use Fast Retransmission

- How to detect packet loss as soon as possible?
  - Problem
    - Timeout period is too long
      - Sender wait long before retransmit lost packet
        - Increase delay
  - Solution
    - Fast retransmission
      - Sender receive triple duplicate ACK for packet *n*
        - Retransmit packet *n*+1



Fast retransmit after sender receive triple duplicate ACK

- Receiver
  - Control a source host's sending rate so that it does not overflow receive host's buffer
  - Inform sender the receive window value **`rwnd`**
    - **`rwnd`**
      - Unit: byte
      - Indicate the buffer space availability at receiver
- Sender
  - Send packets of size < **`rwnd`** bytes to receiver



Receiver's buffer

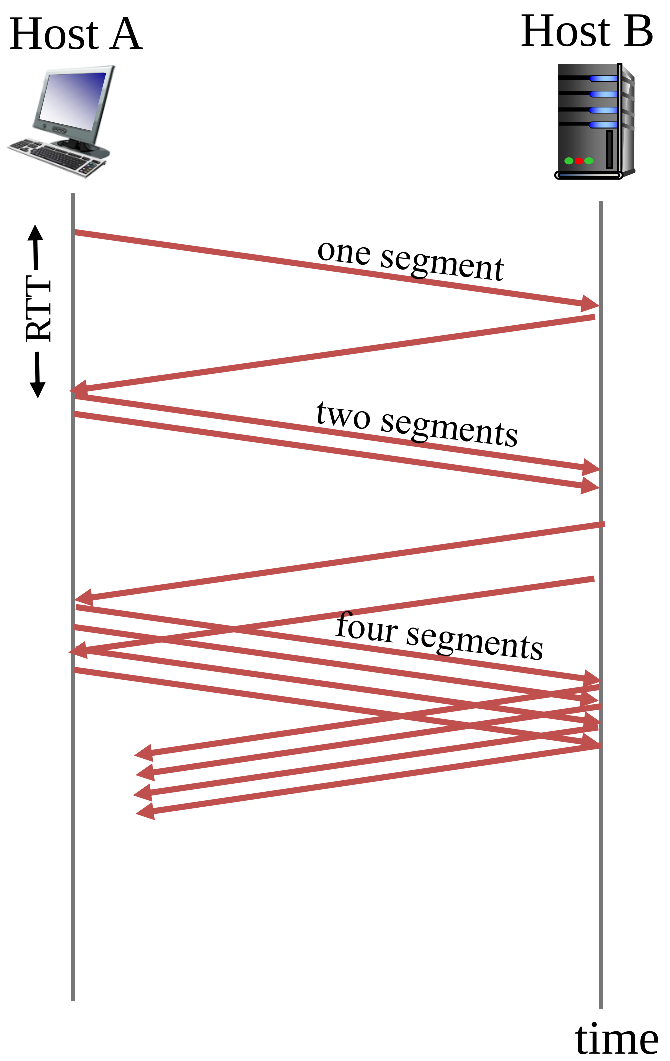
3.7 TCP Congestion Control

SUNWAY UNIVERSITY

- Characteristics
  - End-to-end
    - Destination host infer congestion from observed packet loss and end-to-end delay
      - End-to-end delay = RTT / 2 where RTT is Round Trip Time
- Control a source host's sending rate so that each connection traversing a congested link get an equal share of link bandwidth
  - Sender
    - Send packets of size < min { **`rwnd`** , **`cwnd`** } bytes to receiver
      - **`rwnd`** is receive window
      - **`cwnd`** is congestion window
      - So, either flow control or congestion control take effect
    - Sending rate = **`cwnd`** / RTT bytes/sec
      - This means, sender send **`cwnd`** bytes, wait RTT for ACK, then send more
- Two main components
  - **Slow start**
  - **Congestion avoidance**

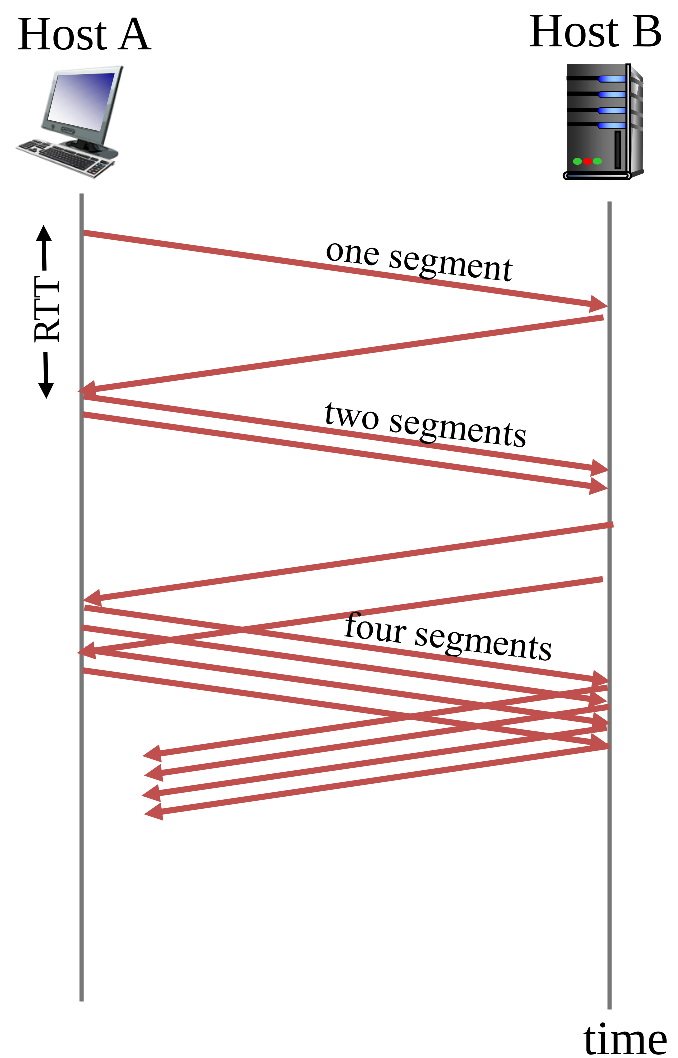
3.7 TCP Congestion Control: Slow Start

- How it work?
  - When connection begin, increase rate exponentially until
    - first packet loss or
    - **`cwnd`** = **`ssthresh`**
      - **`ssthresh`** threshold is reached
  - Increase **`cwnd`** for every ACK received
    - Initially, **`cwnd`** = 1 MSS
      - Maximum Segment Size (MSS) is the maximum amount of data that can be included in a segment
    - Then, double **`cwnd`** every RTT
      - Example:
        - First round: 1 segment
        - Second round: 2 segment
        - Third round: 4 segment
  - So, initial rate is slow but increase exponentially



3.7 TCP Congestion Control: Congestion Avoidance

- How it work?
  - When connection **`cwnd`** = **`ssthresh`**
  - Increase **`cwnd`** by 1 MSS for every ACK received (or 1 RTT)
    - Note: 1 RTT = 1 ACK received
  - So, initial rate is slow but increase exponentially
- During packet loss
  - Firstly, set **`ssthresh`** = **`cwnd`** / 2
  - Then, set **`cwnd`** = 1 MSS



3.7 TCP Congestion Control: Packet Loss

- During packet loss (Either timeout or triple duplicate ACK is received)
  - TCP Tahoe
    - Firstly, set **`ssthresh`** = **`cwnd`** / 2
    - Then, set **`cwnd`** = 1 MSS
  - TCP Reno
    - Firstly, set **`ssthresh`** = **`cwnd`** / 2
    - Then,
      - For triple duplicate ACK
        - set **`cwnd`** = **`cwnd`** / 2
      - For timeout
        - set **`cwnd`** = 1 MSS

- So, TCP congestion control applies **Additive Increase Multiplicative Decrease (AIMD)**
  - Additive increase
    - Increase **`cwnd`** by 1 MSS
  - Multiplicative decrease
    - set **`ssthresh`** = **`cwnd`** / 2