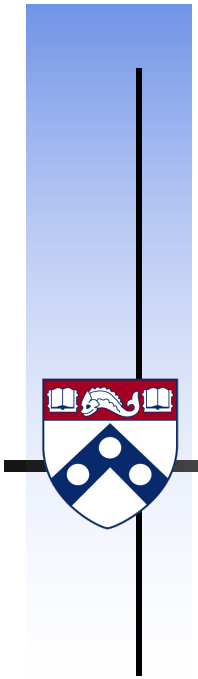


CIS 553: Networked Systems

TCP

March 23, 2020

Course Changes

Lectures: MW at 4:30 pm ET

- Zoom (here) and recording on Canvas

Office hours: https://ohq.io/

Projects

- All projects are now solo
- Scope and difficulty adjusted accordingly

Midterm 2

- Now a take-home exam

Announcements

HW4 Out: Combining HWs 2 and 3

Optional HW2 and 3 Corrections Out

- Grades released later today
- HW2 autograder will be opened
- HW3 test script is sufficient

Optional Midterm 1 Corrections Out

- Raw mean: 56.3 (70.3%); Raw max: 79.5 (99.4%)
- w/o corrections: +24 to your raw score
- w/ corrections: additional 50% of lost points back

Agenda

- Transmission Control Protocol 
  - Stream of bytes service 
  - Reliable in-order delivery 

  - Connection-oriented
  - Flow control
  - Congestion control

Components of a solution

- Checksums (to detect bit errors)
- Acknowledgements (plus retransmissions)
- Sequence numbers (to deal with duplicates)
- Timers (to detect loss)

A Solution: "Stop and Wait"

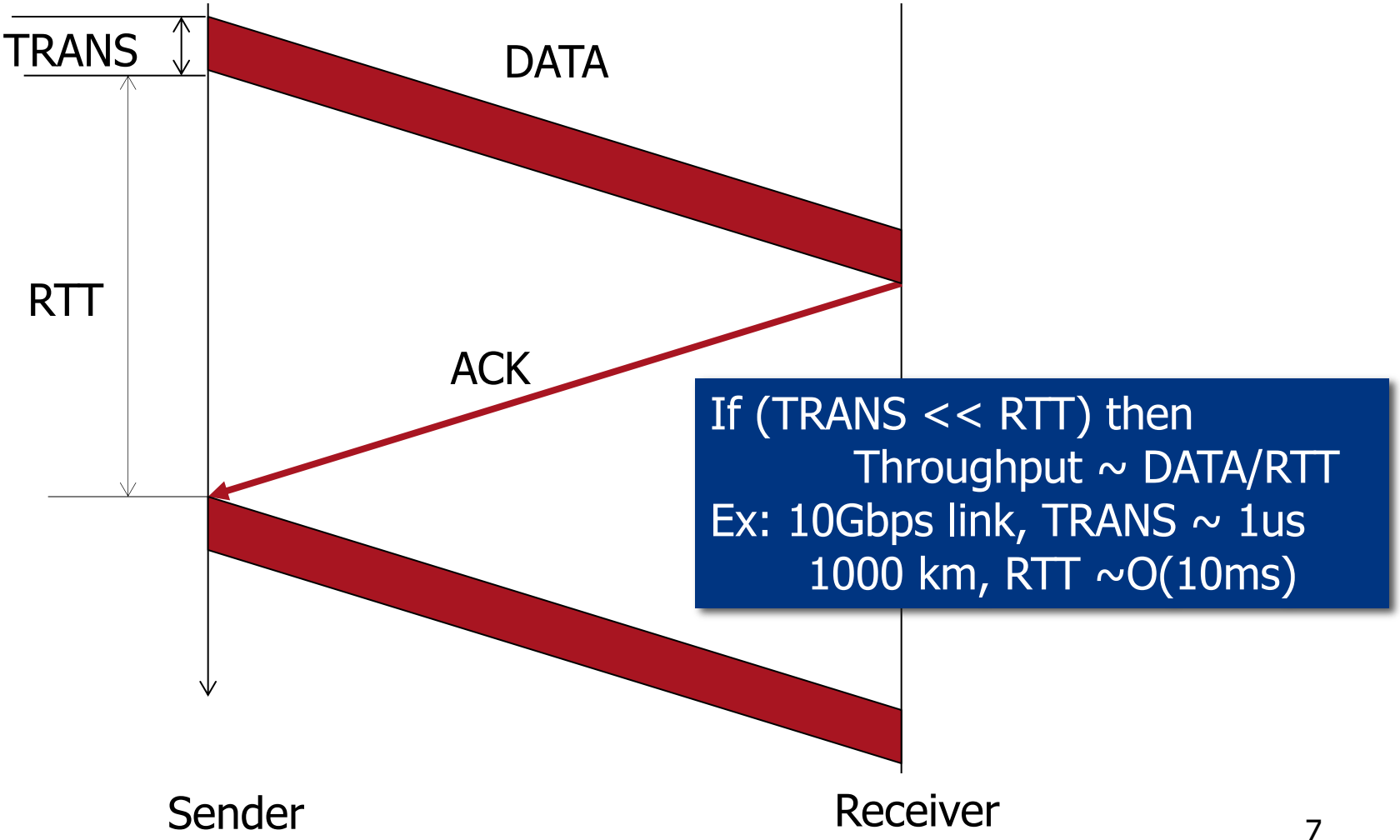
@Sender

- Send packet(I); (re)set timer; wait for ack
- If (ACK)
  - I++; repeat
- If (NACK or TIMEOUT)
  - repeat

@Receiver

- Wait for packet
- If packet is OK, send ACK
- Else, send NACK
- Repeat

Stop and Wait: correct, but inefficient



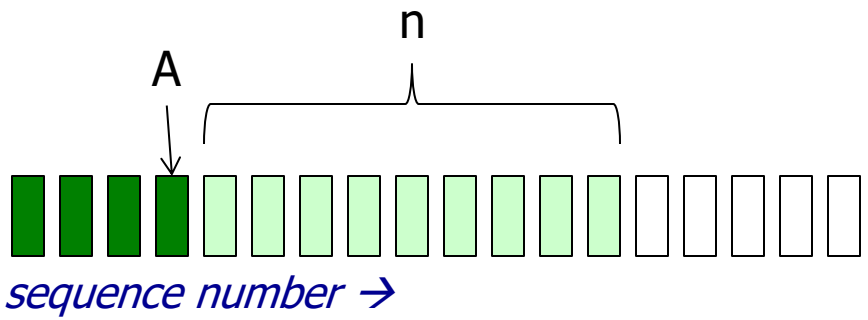
If (TRANS << RTT) then
Throughput ~ DATA/RTT
Ex: 10Gbps link, TRANS ~ 1us
1000 km, RTT ~O(10ms)

Sliding window

- Window = set of adjacent sequence numbers
  - Assume size of window is n
- General idea: send up to n packets at a time
  - Sender can send packets in its window
  - Receiver can accept packets in its window
  - Window of acceptable packets "slides" on successful reception/acknowledgement
  - Window contains all packets that might still be in transit
- Sliding window often called "packets in flight"

Example of sliding window

- Let A be the last ACK received;
  then sender window = {A+1, A+2, …, A+n}

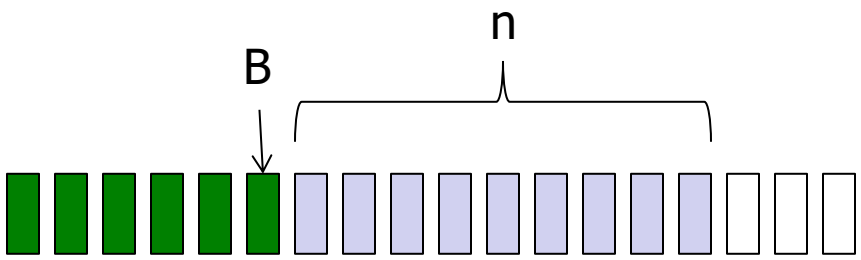


Already ACK'd

Sent but not ACK'd

Cannot be sent

- Let B be the last received packet that we ACK'd;
  then receiver window = {B+1, …, B+n}



Received and ACK'd

Acceptable but not yet received

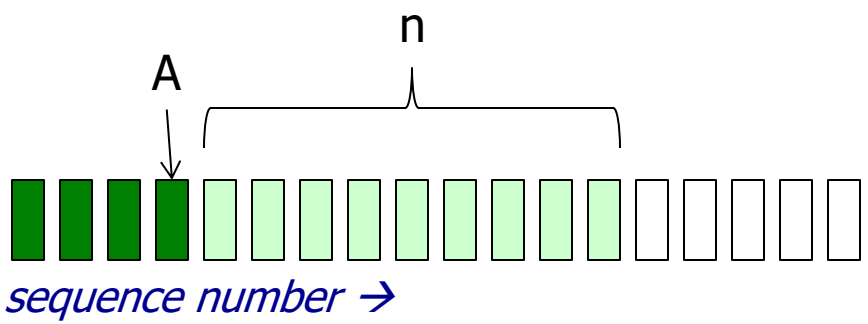
Cannot be received

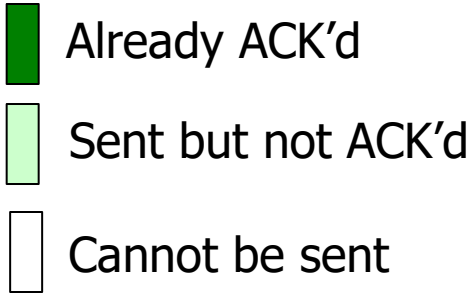
Throughput of sliding window

- If window size is n, then throughput is roughly
  - min(n*DATA/RTT, Link Bandwidth)
- Compare to Stop and Wait: Data/RTT
- Why wouldn't we set n to be very large?

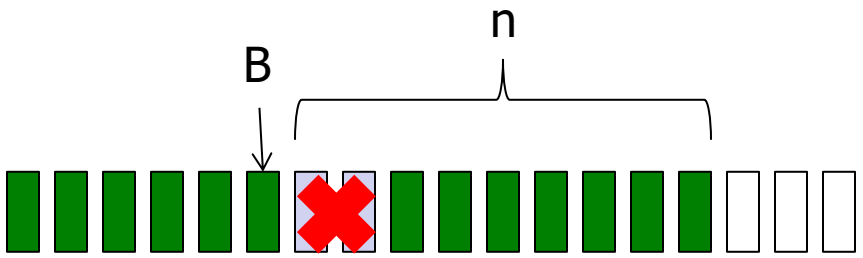
What happens when there is a loss?

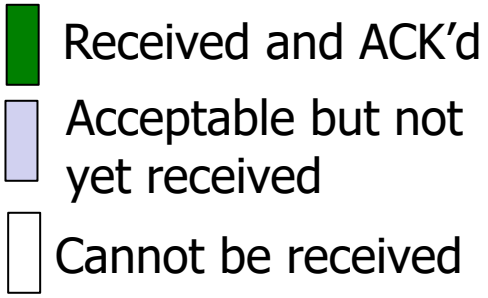
- Let A be the last ACK received;
  then sender window = {A+1, A+2, …, A+n}





- Let B be the last received packet that we ACK'd;
  then receiver window = {B+1, …, B+n}

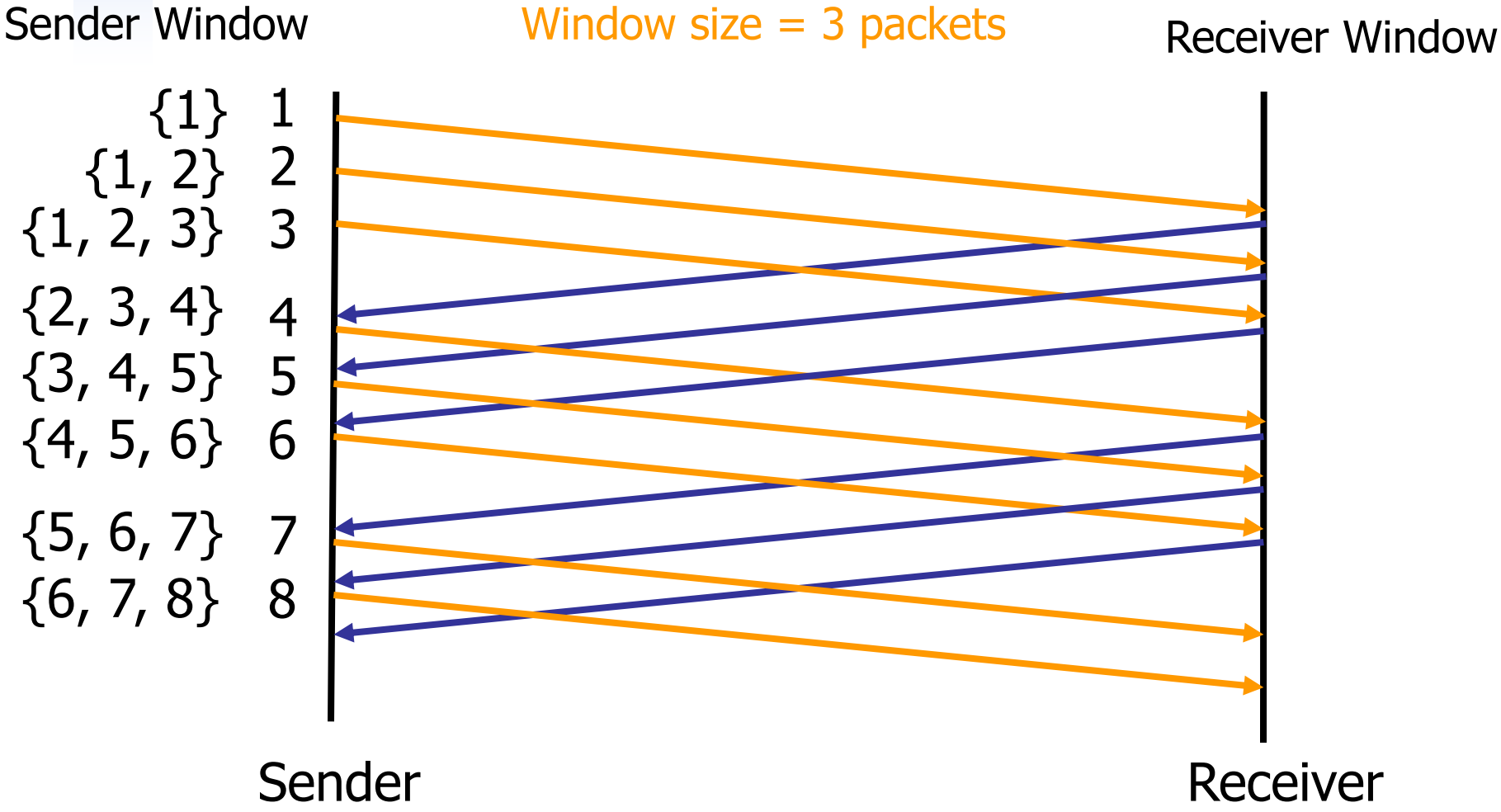




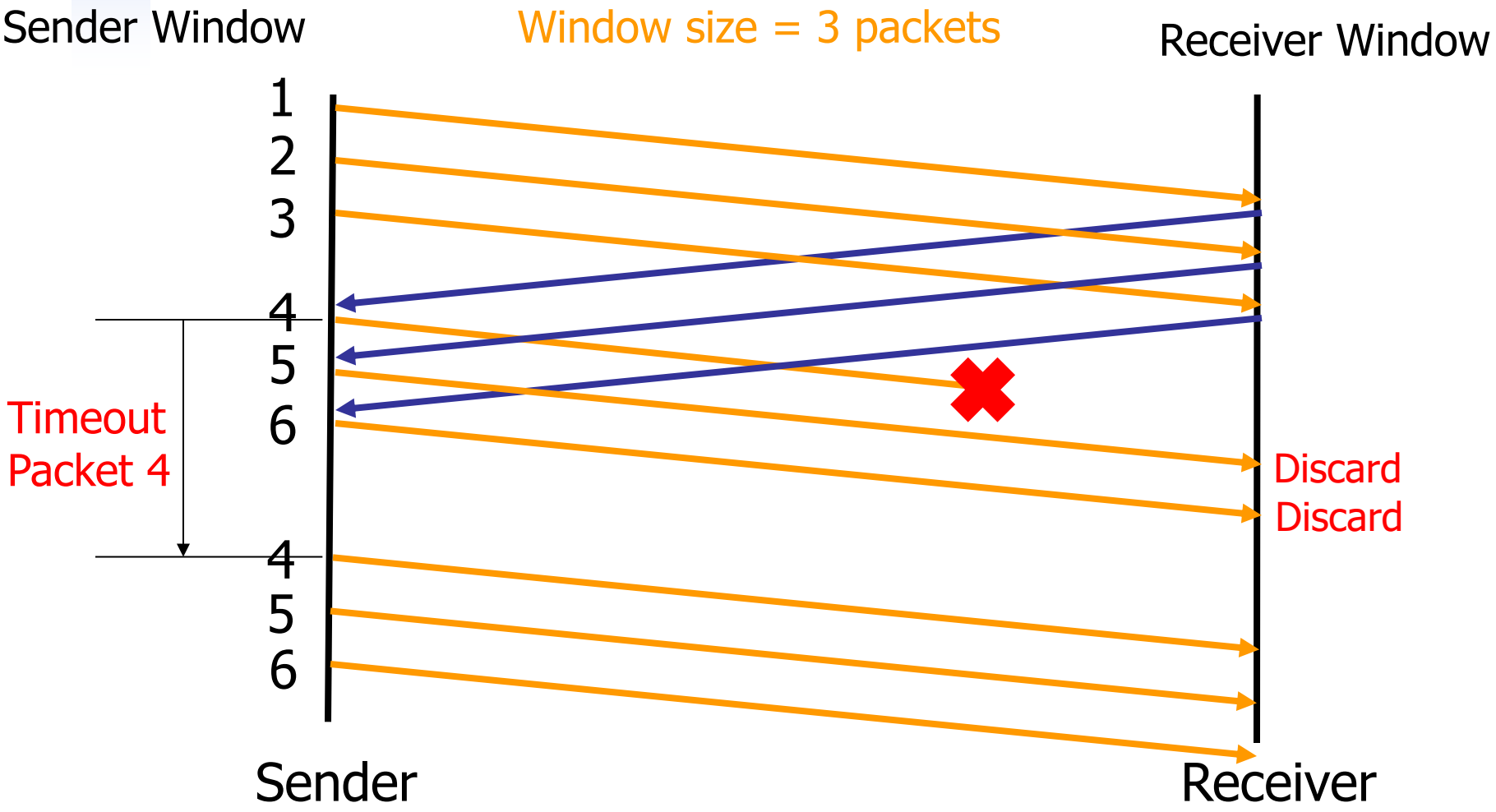
Go-Back-N (GBN)

- Sender transmits up to n unacknowledged packets
- Receiver only accepts packets in order
  - Discards out-of-order packets (i.e., packets other than B+1)
- Receiver uses cumulative acknowledgements
  - i.e., sequence# in ACK = next expected in-order sequence#
- Sender sets timer for 1st outstanding ack (A+1)
- If timeout, retransmit A+1, … , A+n

GBN example w/o errors



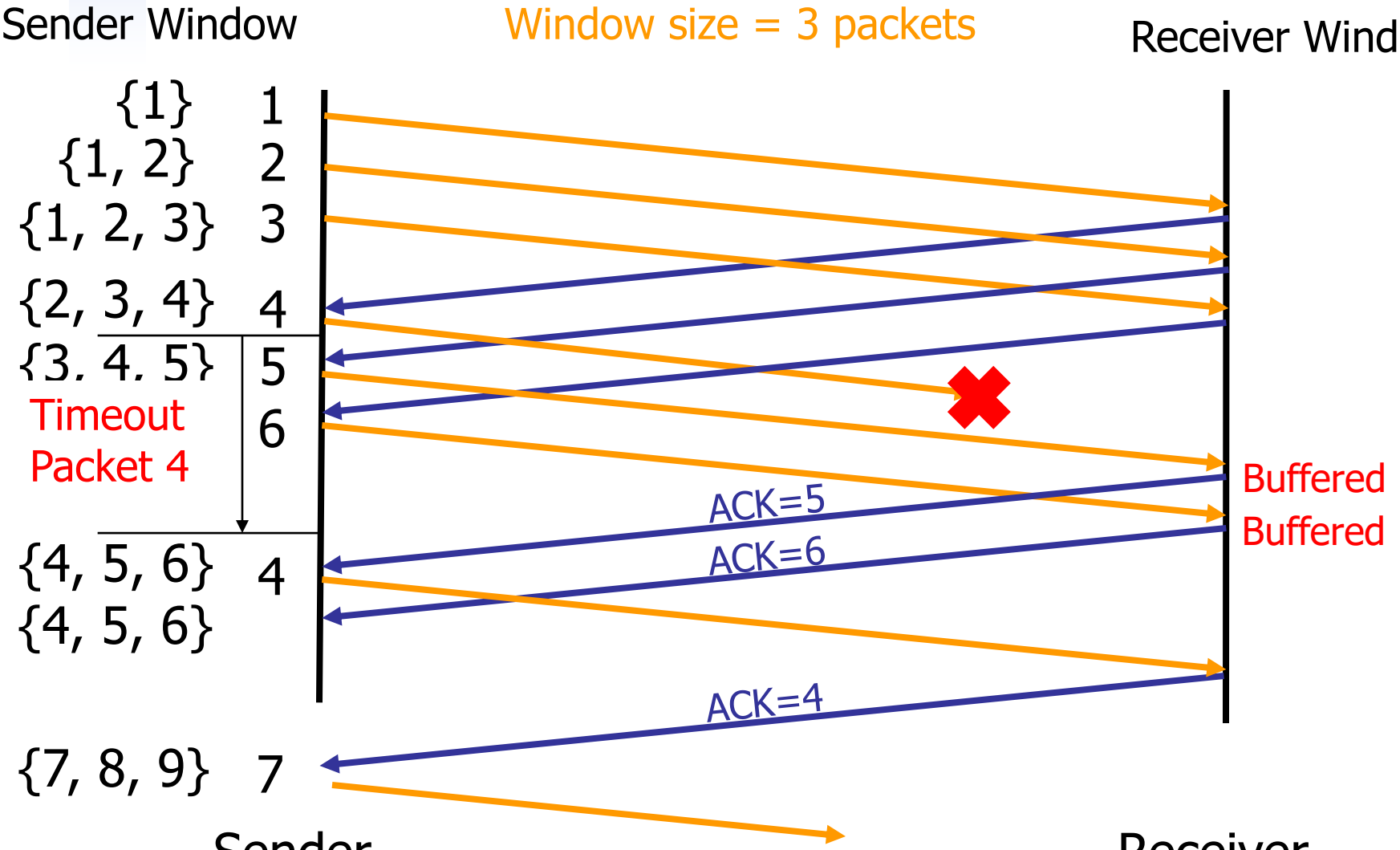
GBN example with errors



Selective Repeat (SR)

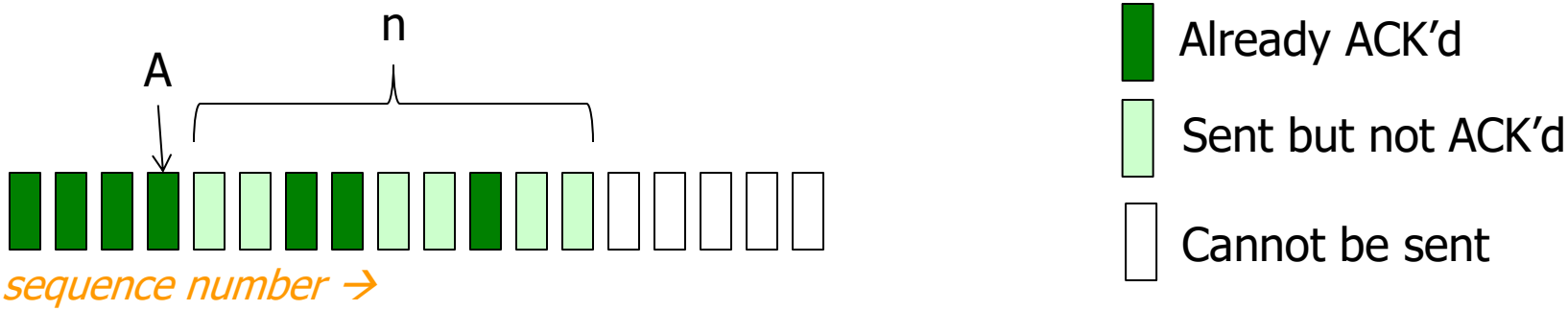
- Sender: transmit up to n unacknowledged packets
- Assume packet k is lost, k+1 is not
  - Receiver: indicates packet k+1 correctly received
  - Sender: retransmit only packet k on timeout
- Efficient in retransmissions, but…
  - Needs complex book-keeping, e.g., timer per packet
  - Needs

SR example with errors

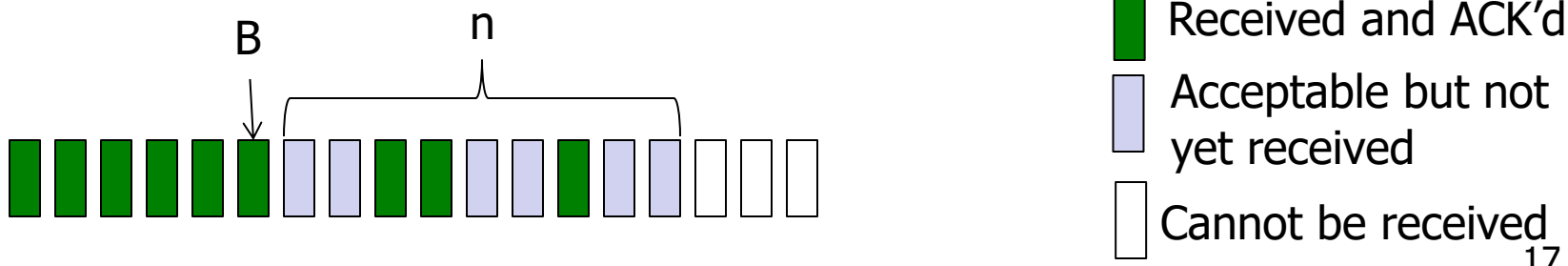


Sliding window w/ SR

- Let A be the last ACK'd packet without gap; then sender window = {A+1, A+2, …, A+n}



- Let B be the last received packet without gap; then receiver window = {B+1, …, B+n}



GBN vs. Selective Repeat

- When would GBN be better?
  - When error rate is low; wastes bandwidth otherwise
- When would SR be better?
  - When error rate is high; wastes memory and adds complexity otherwise

Many more optimizations

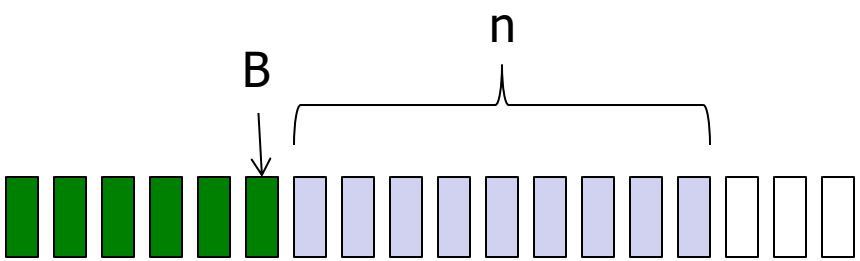
- Checksums (to detect bit errors)
- Acknowledgements (plus retransmissions)
  - Can we avoid using a timeout per packet?
- Sequence numbers (to deal with duplicates)
  - Can we defend against stale packets?
- Timers (to detect loss)
  - How do we set the timer?

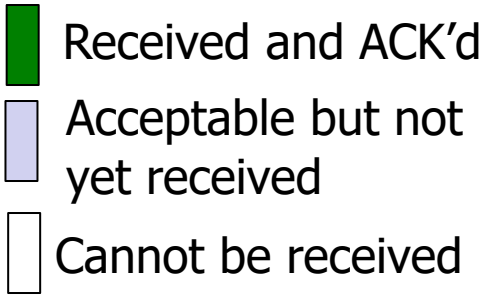
Acknowledgements

- Problem: timeouts are very slow
- Idea: ACKs can carry information beyond just the current packet
  - Cumulative ACKs: ACK carries next in-order sequence number that the receiver expects
  - Selective ACKs: ACK individually acknowledges correctly received packets

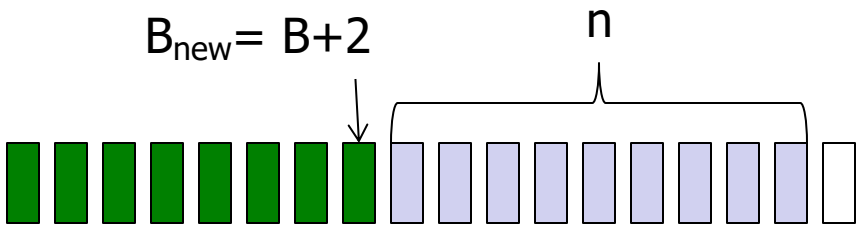
Cumulative acknowledgements

- At receiver





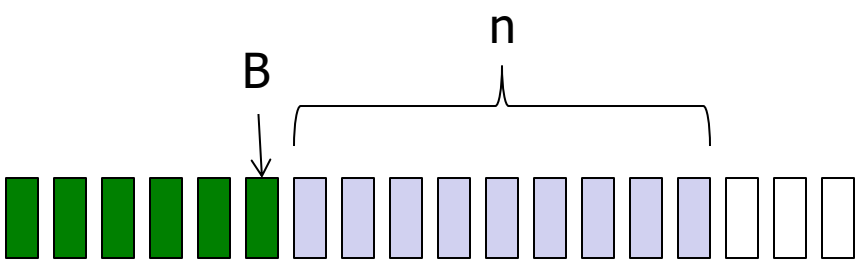
- After receiving B+1, B+2

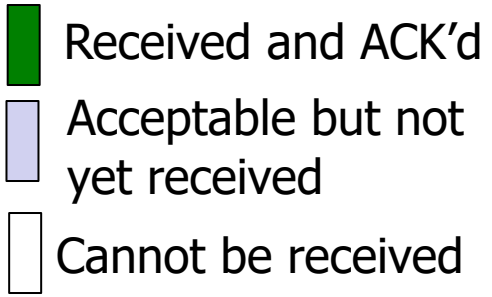


- Receiver sends ACK(B+3) = ACK(B_{new}+1)

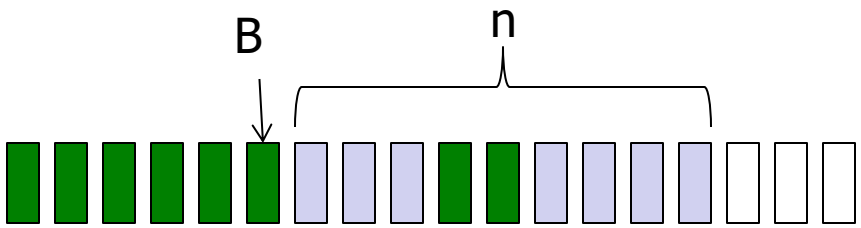
Cumulative acknowledgements (cont'd)

- At receiver





- After receiving B+4, B+5



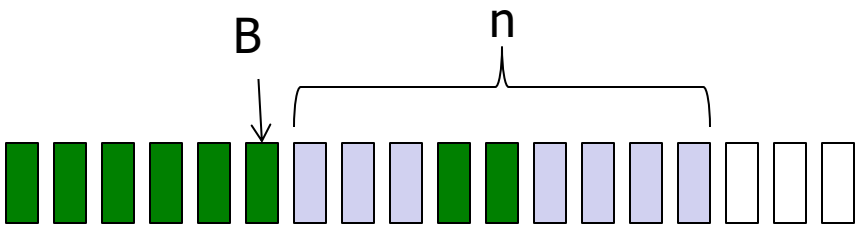
- Receiver sends ACK(B+1)

Taking advantage of cumulative ACKs

- When packet n is lost…
  - … packets n+1, n+2, and so on may get through
- Exploit the ACKs of these packets
  - ACK says receiver is still awaiting nth packet
  - Duplicate ACKs suggest later packets arrived
  - Sender uses "duplicate ACKs" as a hint
- Fast retransmission
  - Retransmit after "triple duplicate ACK"

Selective ACKs

- Send in the ACK the ranges of bytes received
- Example:



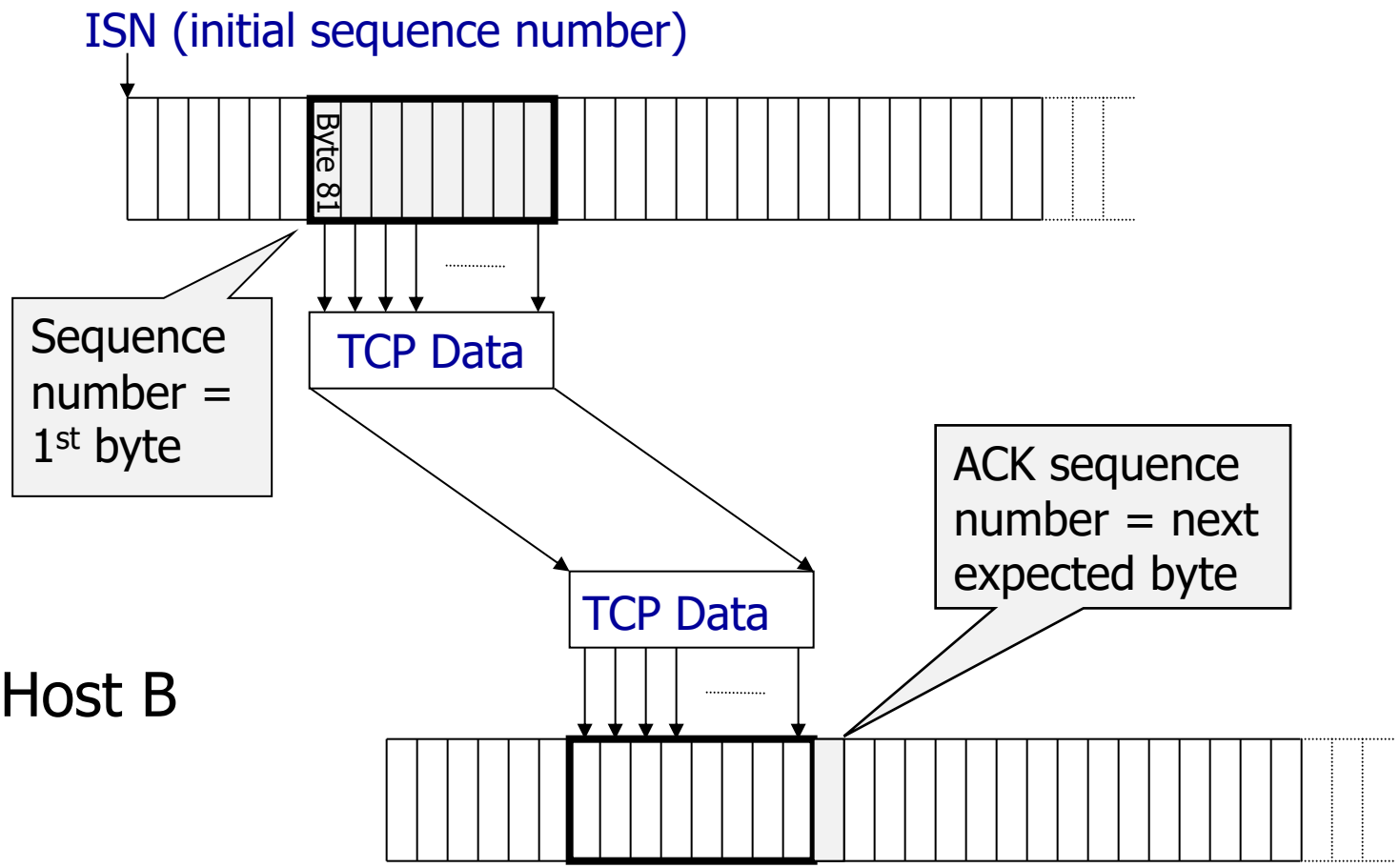
ACK(B+1, [B+4, B+6])

- Selective ACKs offer more precise information but require more complicated book-keeping

Many more optimizations

- Checksums (to detect bit errors)
- Acknowledgements (plus retransmissions)
  - Can we avoid using a timeout per packet?
- Sequence numbers (to deal with duplicates)
  - Can we defend against stale packets?
- Timers (to detect loss)
  - How do we set the timer?

Sequence Number



Initial Sequence Number (ISN)

- Problem: Can't always use ISN of 0
  - Why?
- Reuse of port numbers
  - Port numbers must (eventually) get used again
  - … and an old packet may still be in flight
  - … and associated with the new connection
- Idea: TCP must change the ISN over time
  - Set from a 32-bit clock that ticks every 4 microsec
  - … which wraps around once every 4.55 hours!