Introduction to Computer Networks

TCP Congestion Control (II)

CS640

https://pages.cs.wisc.edu/~mgliu/CS640/S26/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

- Last
  - TCP Congestion Control (I)
- Today
  - TCP Congestion Control (II)
- Announcements
  - Lab 4 due date 04/30/2026

Recap

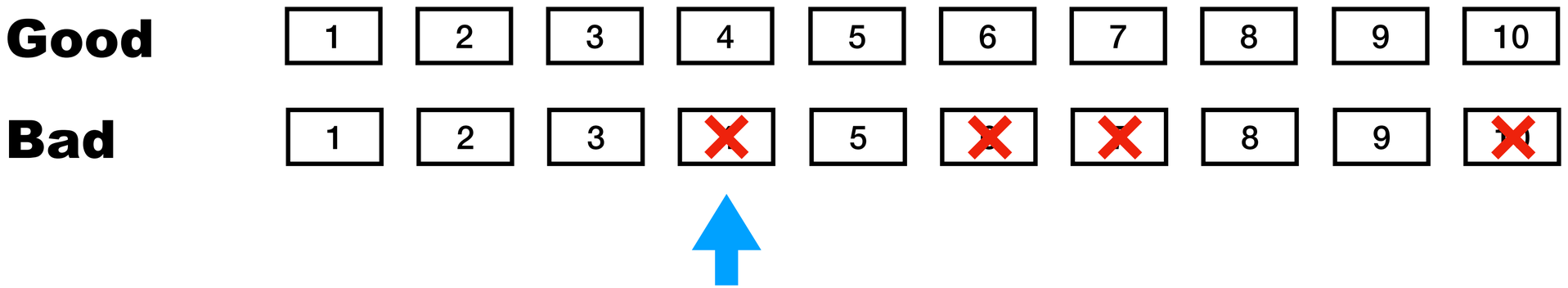
- Key Questions:
  - What is the goal of the TCP congestion control?

- Terminology
  - Congestion window
  - Slow start
  - AIMD

Fast Retransmit

- Use out-of-order ACKs effectively

- Three duplicated ACKs
  - Resend the same acknowledgment to the first missing segment
  - Streamline the implementation



Fast Recovery

- Slow start probing is unnecessary under duplicated ACKs

- Adjust the Congestion Window to the Congestion Threshold
  - Congestion window = congestion threshold

Improve the Congestion Control “Round”

- What is round?
  - A round is the time it takes to send all data within the congestion window and receive the corresponding ACKs
  - So round is a dynamic epoch

- Make congestion control to react on each ACK
  - Reacting at the round granularity is slow
  - An ACK indicates there is room to send data

Combine Everything Together

- Congestion control is a window adjustment algorithm
  - #1: Reaction point (RP) or sender
  - #2: Congestion point (CP) or switch/router
  - #3: Notification Point (NP) or receiver

Combine Everything Together

- Congestion control is a window adjustment algorithm
  - #1: Reaction point (RP) or sender
  - #2: Congestion point (CP) or switch/router
  - #3: Notification Point (NP) or receiver

} Adjust window

} Issue implicit feedbacks

Combine Everything Together

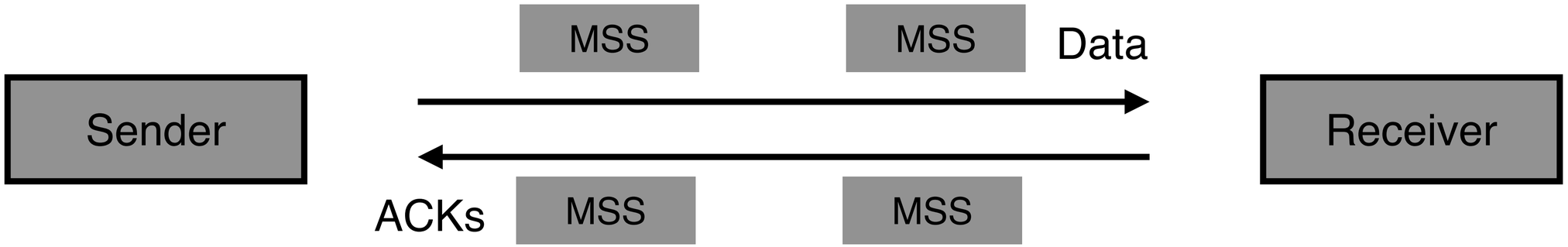
- Congestion control is a window adjustment algorithm
  - #1: Reaction point (RP) or sender
  - #2: Congestion point (CP) or switch/router
  - #3: Notification Point (NP) or receiver

} Adjust window

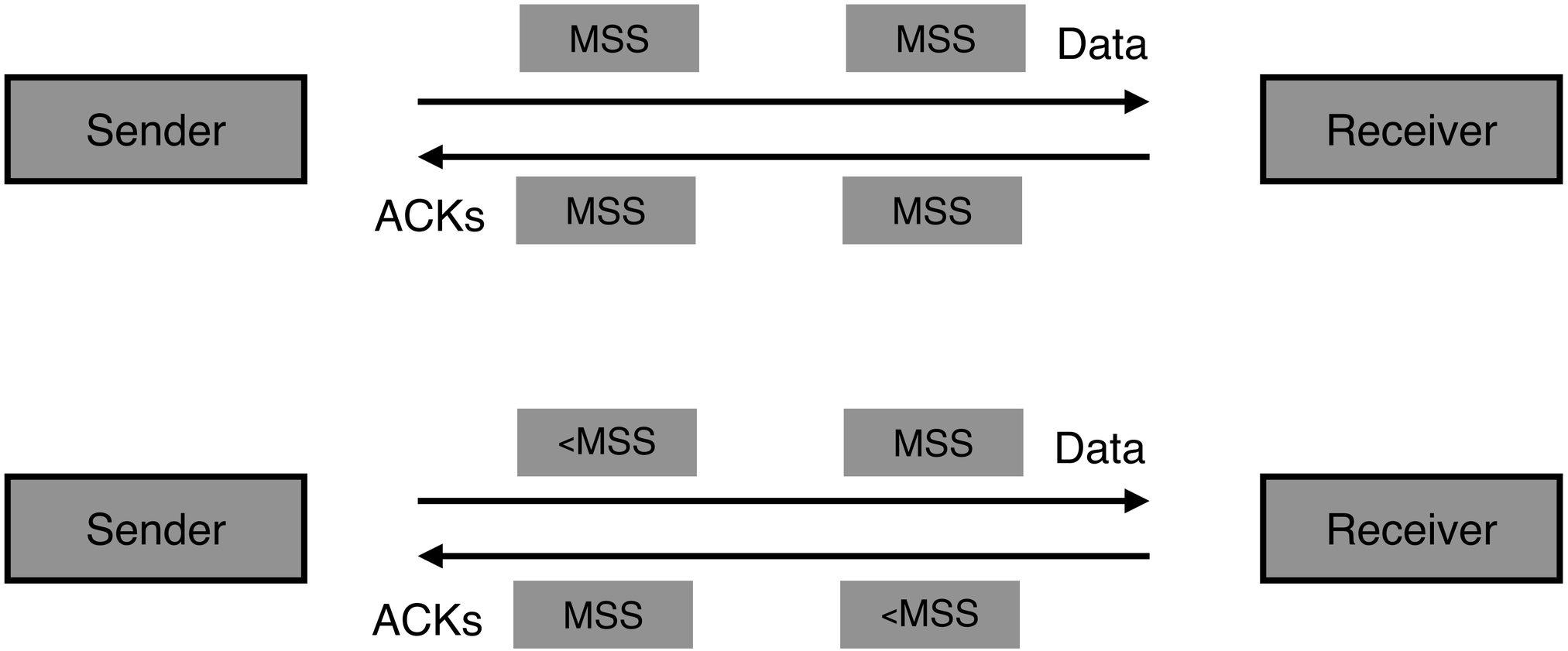
} Issue implicit feedbacks

- TCP Reno
  - One of many congestion control algorithms
  - #1: Slow start
  - #2: AIMD
  - #3: Fast retransmit/recovery
  - #4: Per-ACK adjustment

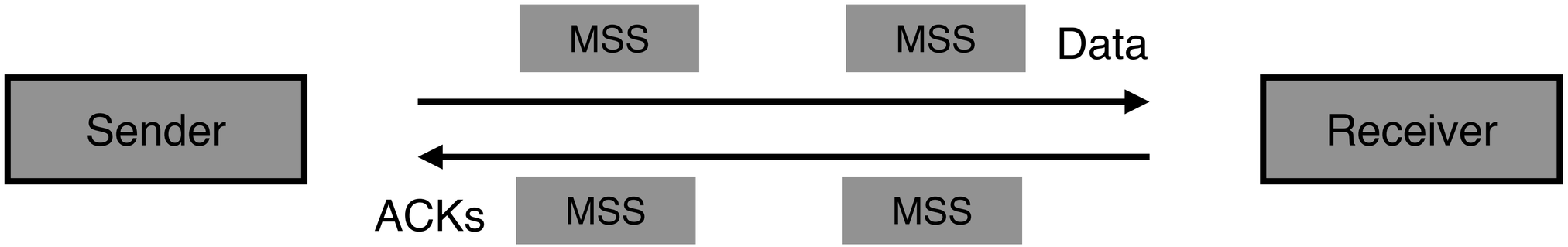
Issue #1: Silly Window Syndrome



Issue #1: Silly Window Syndrome



Issue #1: Silly Window Syndrome



Problem:

- **Wait too long, hurt latency**
- **Wait too short, hurt bandwidth**

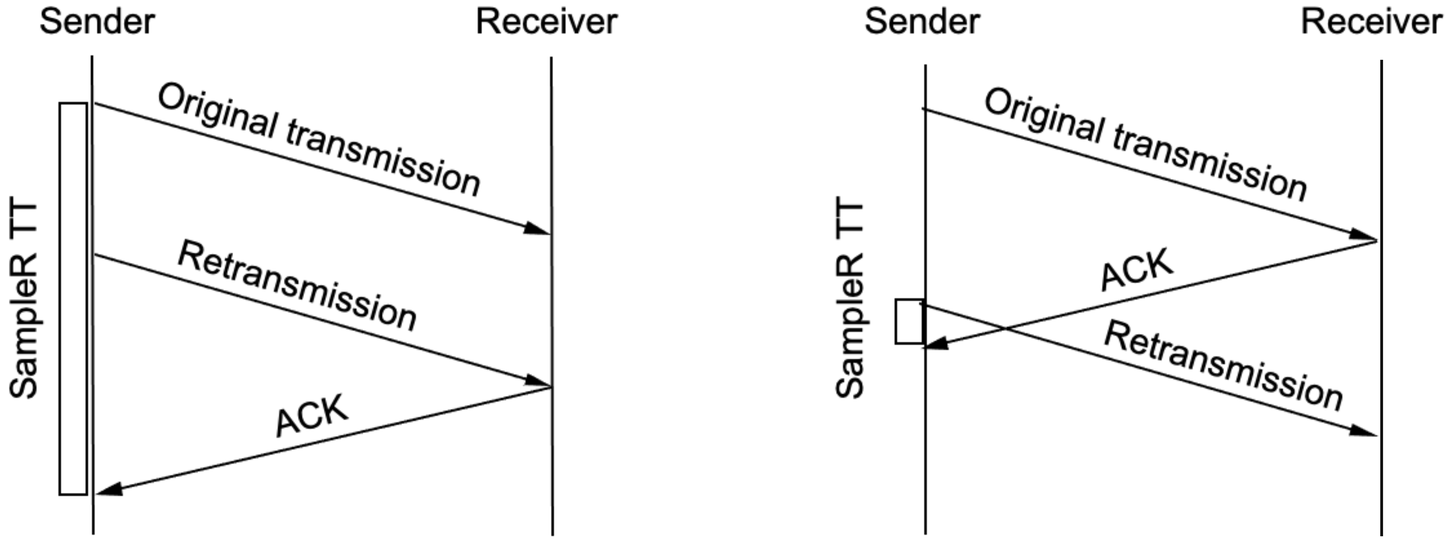
Solution: Nagle's Algorithm

- A self-clocking solution
  - As long as TCP has any data in flight, the sender will eventually receive an ACK
  - TCP_NODELAY option

```
When the application produces data to send
  if both the available data and the window ≥ MSS
    send a full segment
  else
    if there is unACKed data in flight
      buffer the new data until an ACK arrives
    else
      send all the new data now
```

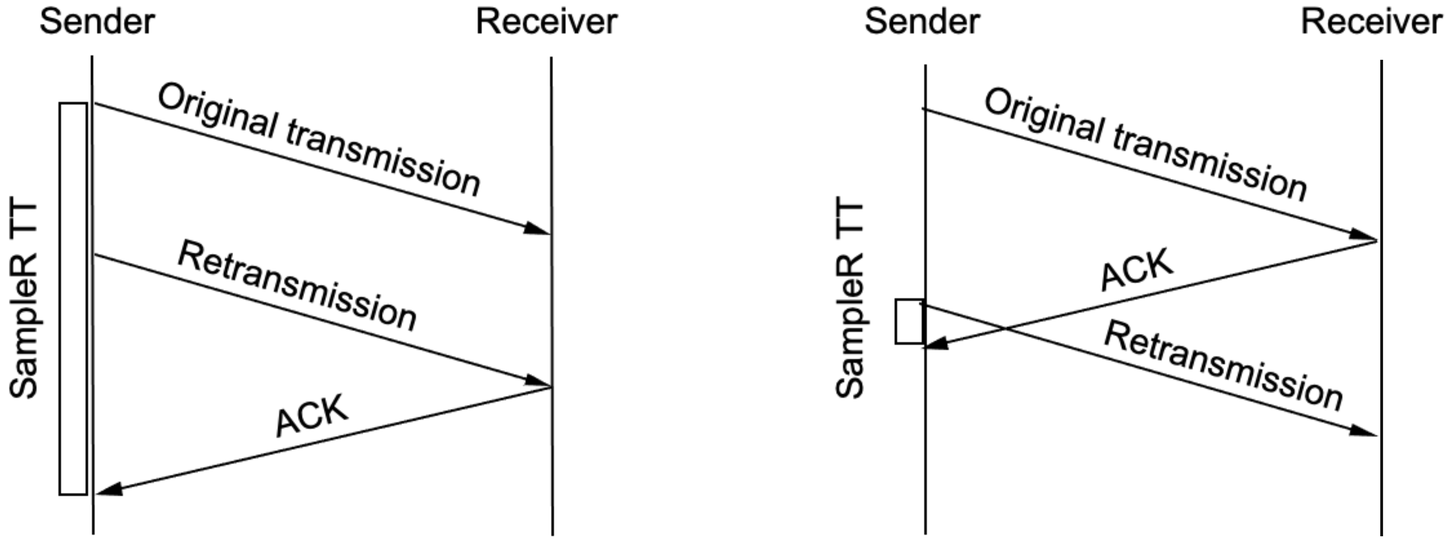
Issue #2: Timeout Setup during Retransmission

- Degenerate case
  - Do not sample RTT when retransmitting



Karn/Partridge Algorithm for RTO

- Set the next RTO to be 2x RTO_last after each retransmission
  - Exponential backoff is a well-known control theory method
  - Loss is mostly likely caused by congestion



Issue #3: Retransmitted Segments

- What segments are retransmitted under a timeout?
  - Option #1: all segments subsequently after the missing one (pessimistic)
  - Option #2: just the missing one (optimistic)

Issue #3: Retransmitted Segments

- What segments are retransmitted under a timeout?
  - Option #1: all segments subsequently after the missing one (pessimistic)
  - Option #2: just the missing one (optimistic)

- Solution: selective acknowledgment
  - The receiver uses optional fields to acknowledge the missing ones
  - SACK option

Issue #3: Retransmitted Segments

- What segments are retransmitted under a timeout?
  - Option #1: all segments subsequently after the missing one (pessimistic)
  - Option #2: just the missing one (optimistic)

- Selective acknowledgment
  - The receiver uses optional fields to acknowledge the missing ones
  - SACK option

Tell the sender what segments have been arrived

TCP SACK

- Same congestion control mechanisms as TCP Reno
  - Uses TCP options fields
  - Timeouts are still used
- Tell the sender which segments are received out of order
  - Enable the sender to maintain an image of the receiver’s queue
- The sender resends all missing segments without a timeout
  - Don’t send beyond the congestion window
  - Send new data when no old data needs to be resent

Summary

- Today
  - TCP congestion control (II)

- Next lecture
  - TCP congestion control (III)