Introduction to Computer Networks

CS640

TCP Congestion Control (II)

https://pages.cs.wisc.edu/~mgliu/CS640/S25/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

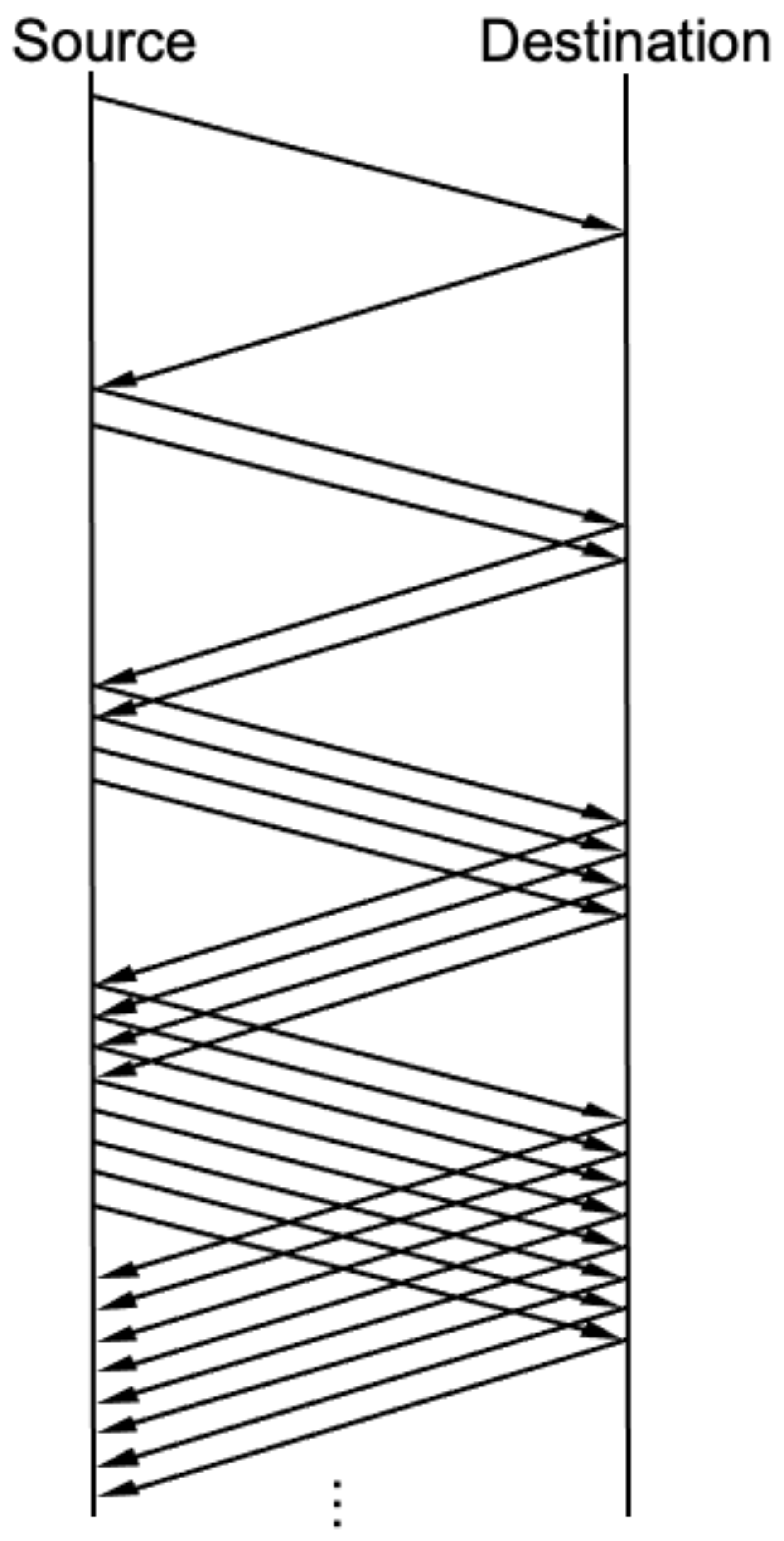
- Last
  - TCP Congestion Control (I)
- Today
  - TCP Congestion Control (II)
- Announcements
  - No class this Thursday (04/17)
  - Lab 4 due date 05/01/2025 12:01PM

Recap: UDP Issues

- **#1: Arbitrary communication**
  - **Senders and receivers can talk to each other in any ways**
- #2: No reliability guarantee
  - Packets can be lost/duplicated/reordered during transmission
  - A checksum is not enough
- #3: No resource management
  - Each channel works as an exclusive network resource owner
  - No adaptive support for the physical networks and applications

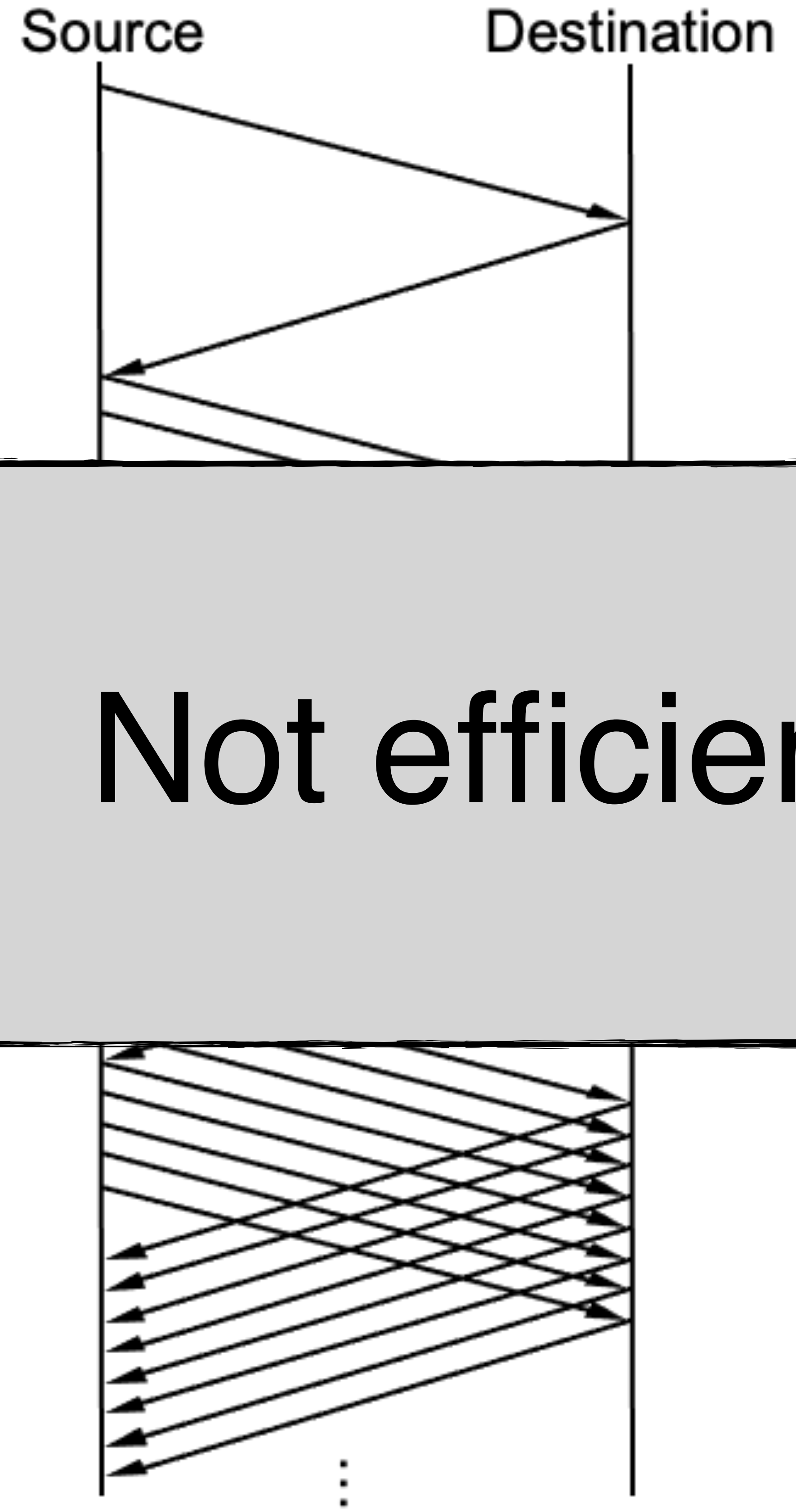
TCP Slow Start

- Determine the available networking capacity exponentially
  - At round i, probe the congestion window with 2^(i-1) segments



TCP Slow Start

- Determine the available networking capacity exponentially
  - At round i, probe the congestion window with 2^(i-1) segments



Not efficient

The goal of TCP congestion control:

Effectively use the networking resources

- Utilization: each networking hardware is fully utilized
- Fairness: each networking hardware is equally shared

Running Phase Bandwidth Adjustment: AIMD

- Adjust the window for fairness and high utilization

- Additive Increase/Multiplicative Decrease
  - Additive increase CongestionWindow when the congestion goes down
  - Multiplicate decrease CongestionWindow when the congestion goes up

Congestion goes up

- Signals:
  - Packet loss
  - Out-of-order ACK

Congestion goes up

- Signals:
  - Packet loss
  - Out-of-order ACK
- Multiplicative Decrease and Why

Congestion goes up

- Signals:
  - Packet loss
  - Out-of-order ACK
- Multiplicative Decrease and Why
  - Quickly shrink the congestion window to avoid congestion collapse
  - Reduce to 1 segment under heavy contention => packet loss
  - Reduce to >1 segment under light contention => out-of-order ACK
  - Congestion Window = Congestion Window X Parameter

Congestion goes down

- Signals:
  - Packets from the last congestion window are delivered successfully

Congestion goes down

- Signals:
  - Packets from the last congestion window are delivered successfully

- Additive Increase and Why

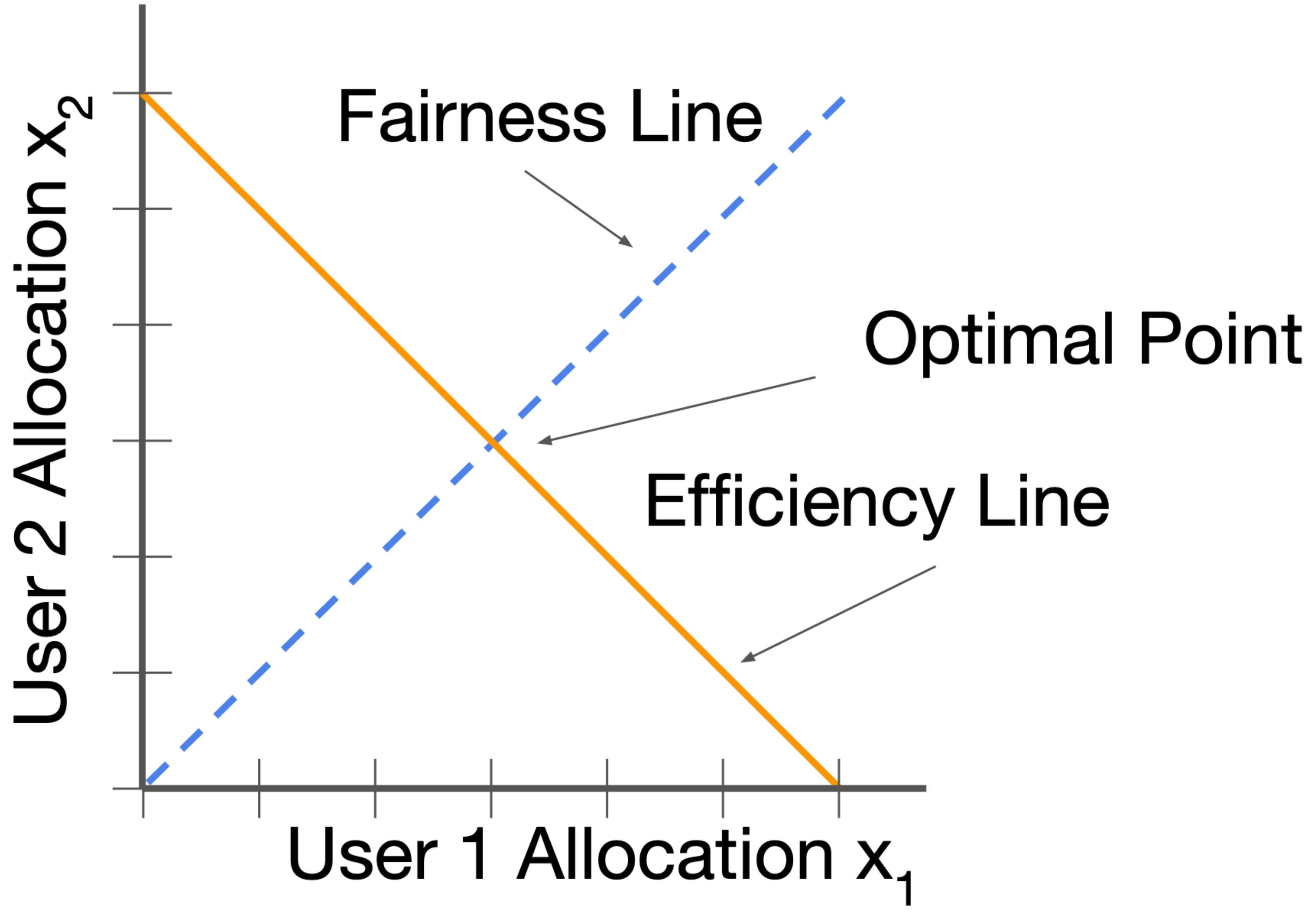
Congestion goes down

- Signals:
  - Packets from the last congestion window are delivered successfully

- Additive Increase and Why
  - Gradually approach the equal bandwidth share offered by the network
  - The addition enables fairness guarantees implicitly
  - No exponential increase since the slow start has found the max in
  - Congestion Window = Congestion Window + Parameter

Discussion

- Parameter selection is hard
  - People sometimes use the fluid model to find out the optimal ones
- The congestion control should converge to the optimal point.
  - AIMD can, but takes several rounds.



Some References

- An ideal AIMD algorithm
  - Efficiency, fairness, convergence, and distributeness

Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks

Dah-Ming CHIU and Raj JAIN
Digital Equipment Corporation, 550 King Street (LKG1-2/A19), Littleton, MA 01460-1289, U.S.A.

New Address: Raj Jain, Washington University in Saint Louis, jain@cse.wustl.edu, http://www.cse.wustl.edu/~jain

Abstract. Congestion avoidance mechanisms allow a network to operate in the optimal region of low delay and high throughput, thereby, preventing the network from becoming congested. This is different from the traditional congestion control mechanisms that allow the network to recover from the

1. Introduction

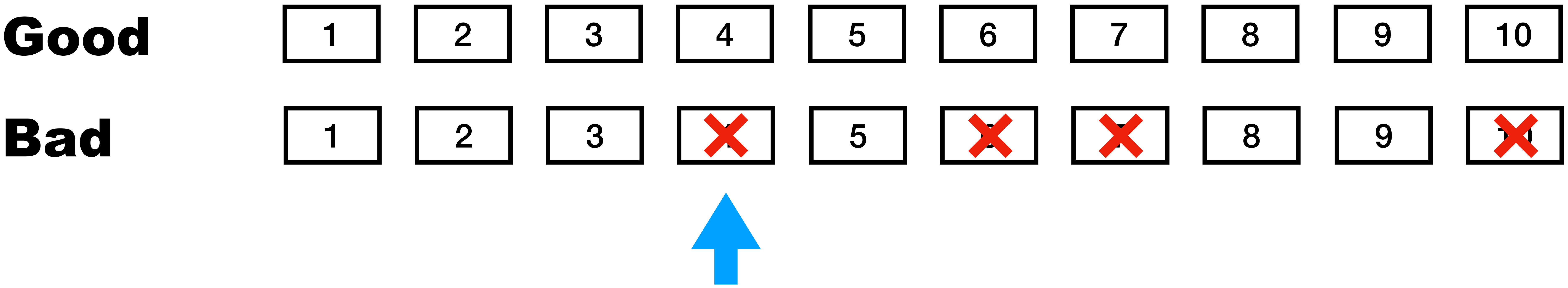
1.1. Background

Congestion in computer networks is becoming an important issue due to the increasing mismatch in link speeds caused by intermixing of old and new technology. Recent technological advances

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

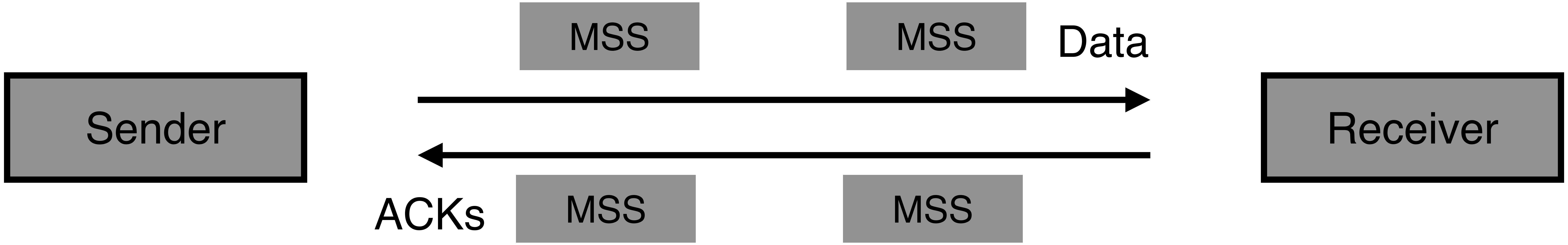
} Issue implicit feedbacks

Combine Everything Together

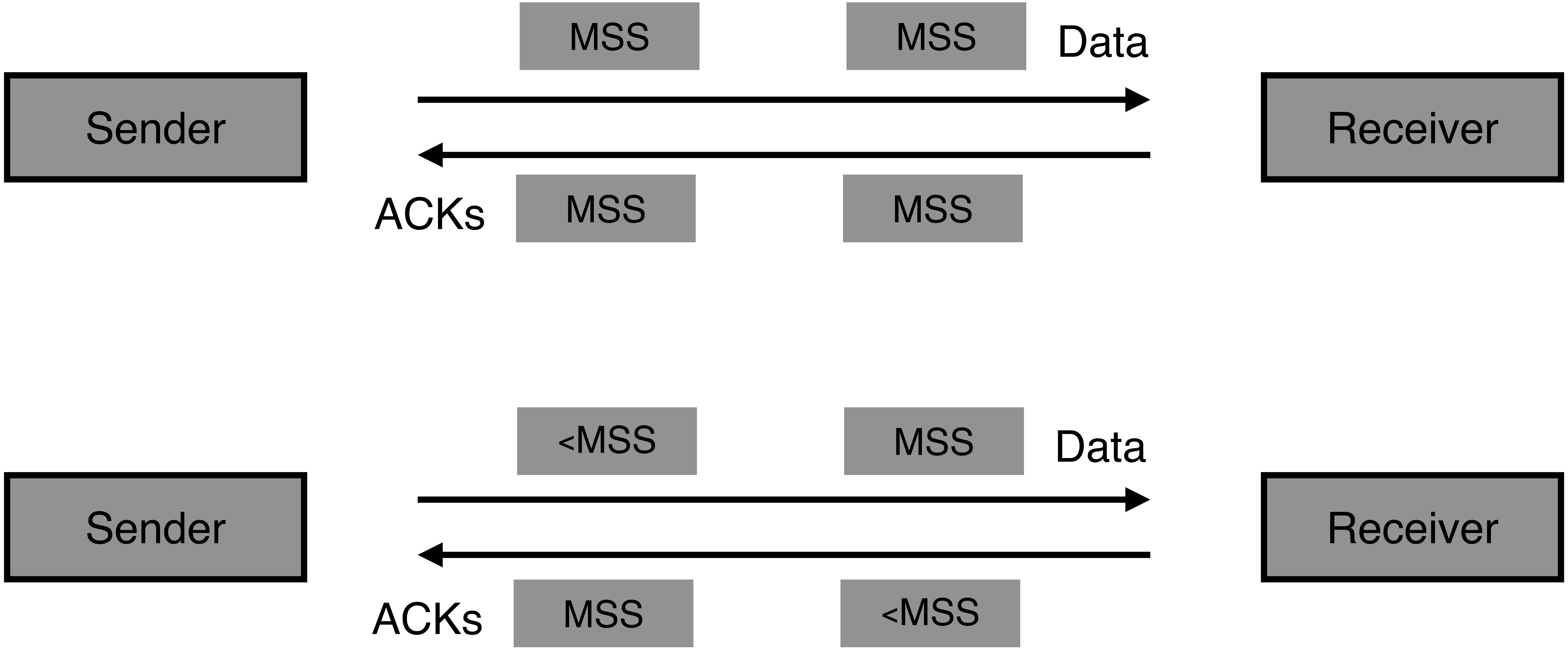
- Congestion control is a window adjustment algorithm
  - #1: Reaction point (RP) or sender } Adjust window
  - #2: Congestion point (CP) or switch/router
  - #3: Notification Point (NP) or receiver } Issue implicit feedbacks

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

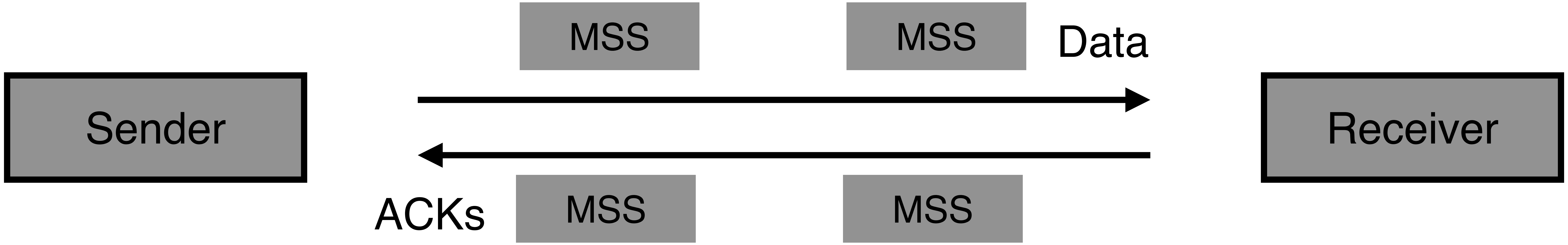
Issue #1: Silly Window Syndrome



Issue #1: Silly Window Syndrome



Issue #1: Silly Window Syndrome



Problem:

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

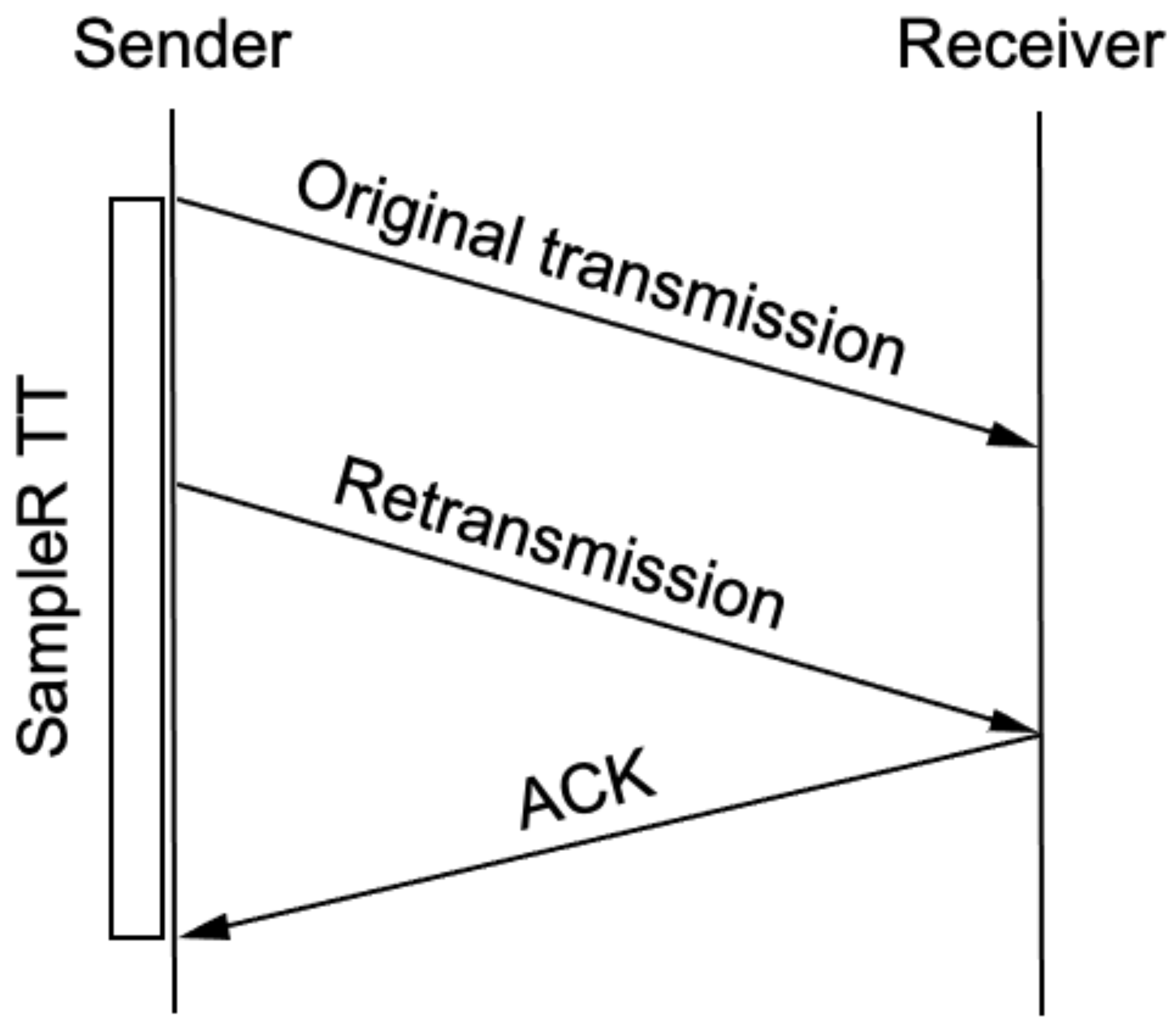
Solution: Nagle's Algorithm

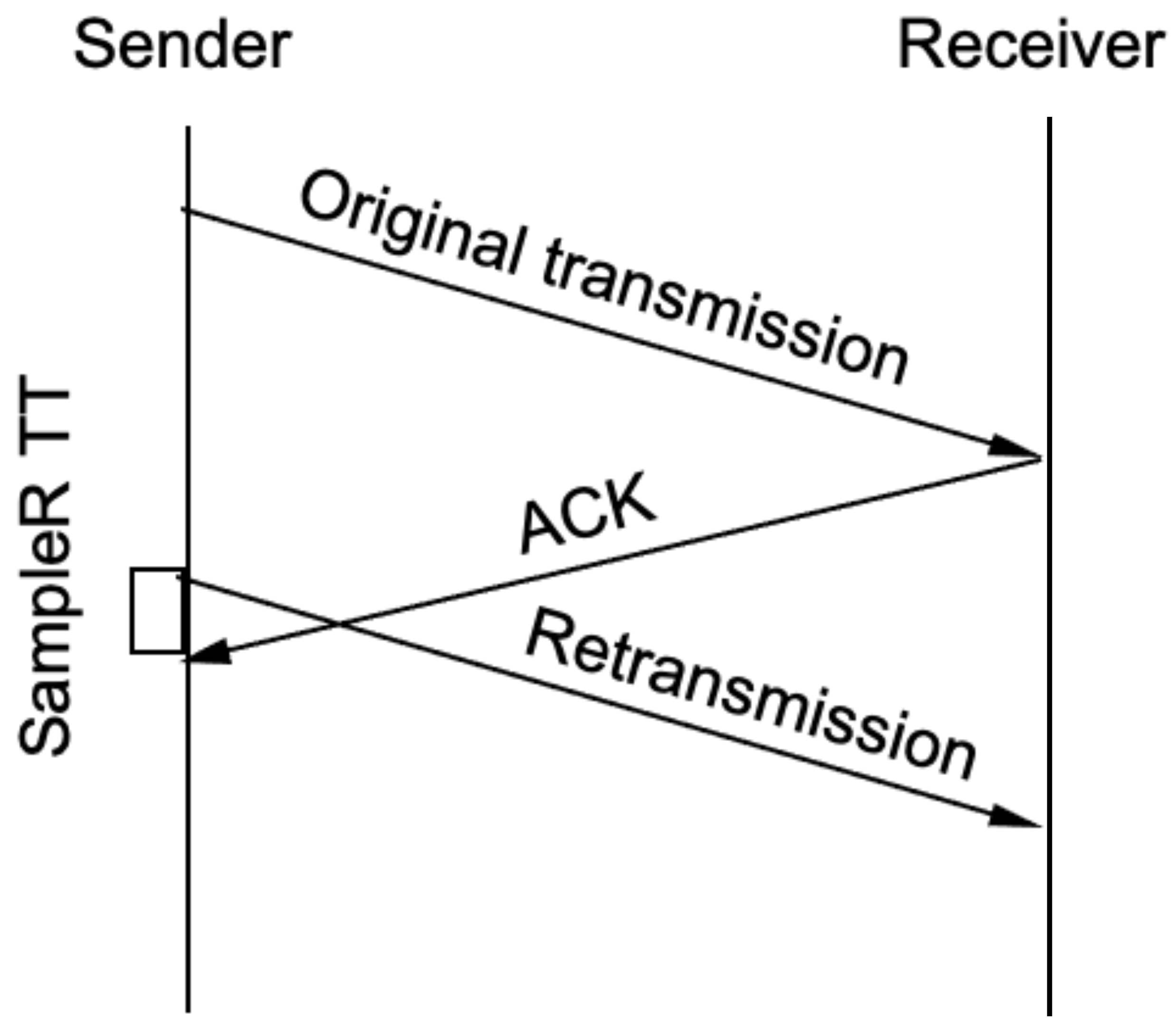
- A self-clocking solution
  - As long as TCP has any data in flight, the sender will eventually receive an ACK
  - TCP_NODELAY option

```
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

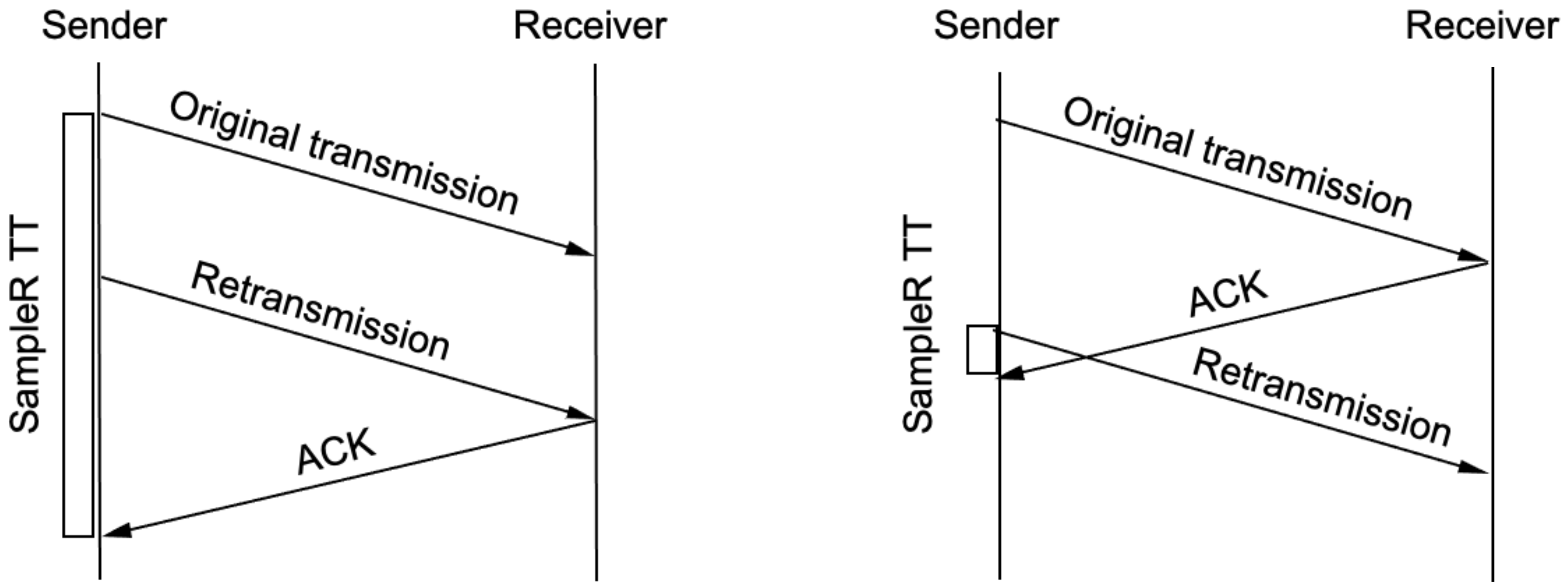
- Degenerate case
  - Do not sample RTT when retransmitting





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 upon out-of-order
  - Enable the sender to maintain an image of the receiver's queue
- The sender resends all missing segments without timeout
  - Don't send beyond the congestion window
  - Send new data when no old data needs to be resent

How does TCP solve the third issue?

- #1: Arbitrary communication
  - Senders and receivers can talk to each other in any ways
- #2: No reliability guarantee
  - Packets can be lost/duplicated/reordered during transmission
  - A checksum is not enough
- **#3: No resource management**
  - **Each channel works as an exclusive network resource owner**
  - **No adaptive support for the physical networks and applications** ✓

Summary

- Today
  - TCP congestion control (II)

- Next lecture
  - TCP in-network support