Introduction to Computer Networks

CS640

TCP Congestion Control (II)

https://pages.cs.wisc.edu/~mgliu/CS640/F22/

Ming Liu
mgliu@cs.wisc.edu

Today

Last lecture

- How to share networking bandwidth among concurrent TCP flows?

Today

- How to improve the efficiency of TCP congestion control?

Announcements

- Lab4 is due 12/02/2022, 11:59 PM
- Lab5 is due 12/14/2022, 11:59 PM
- Final exam: Dec 17, 2022 5:05 PM – 7:05 PM

How TCP solves the first 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
- Checksum is not enough

#3: No resource management

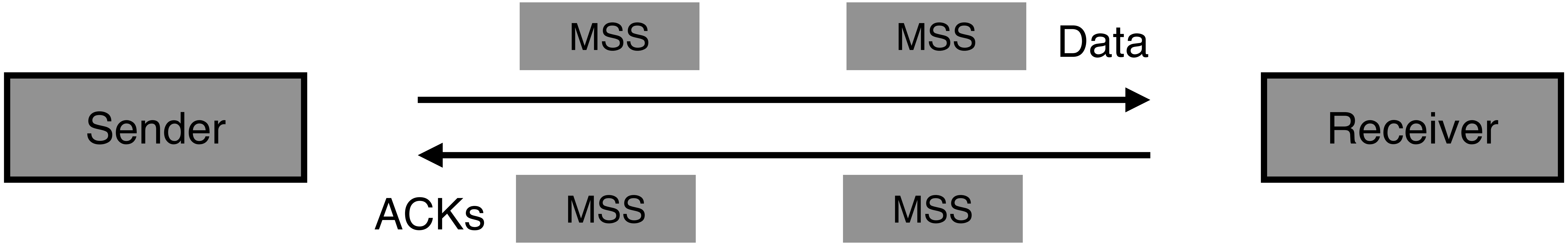
- Each communication channel works as an exclusive network resource owner
- No adaptiveness support for the physical networks and applications

Q: What techniques does TCP Reno introduce?

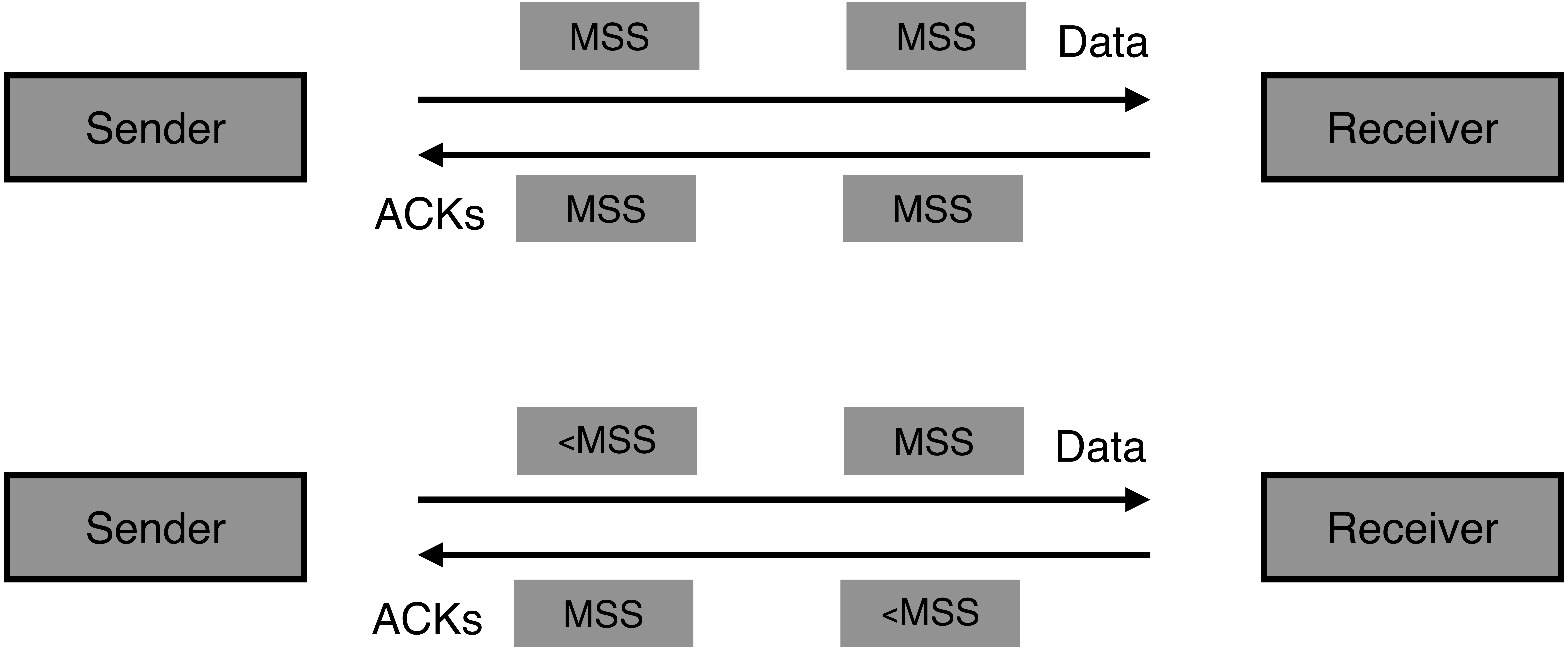
A: Three techniques:

- #1: AIMD
- #2: Slow start
- #3: Fast retransmit and recovery

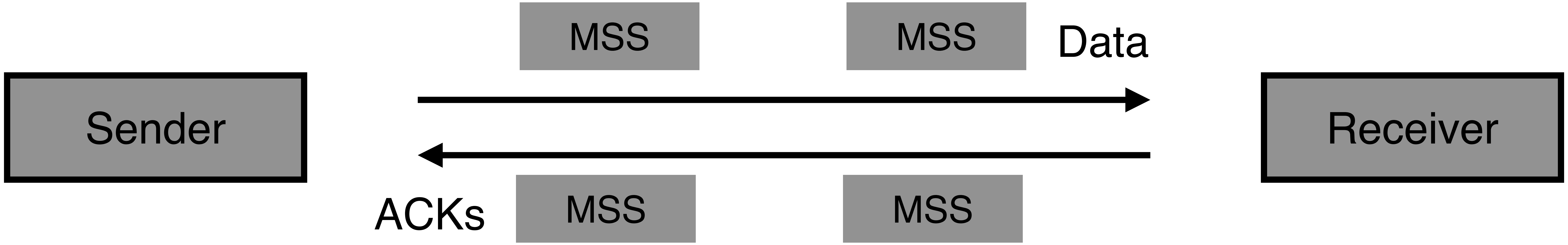
Issue #1: Silly Window Syndrome



Issue #1: Silly Window Syndrome



Issue #1: Silly Window Syndrome



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

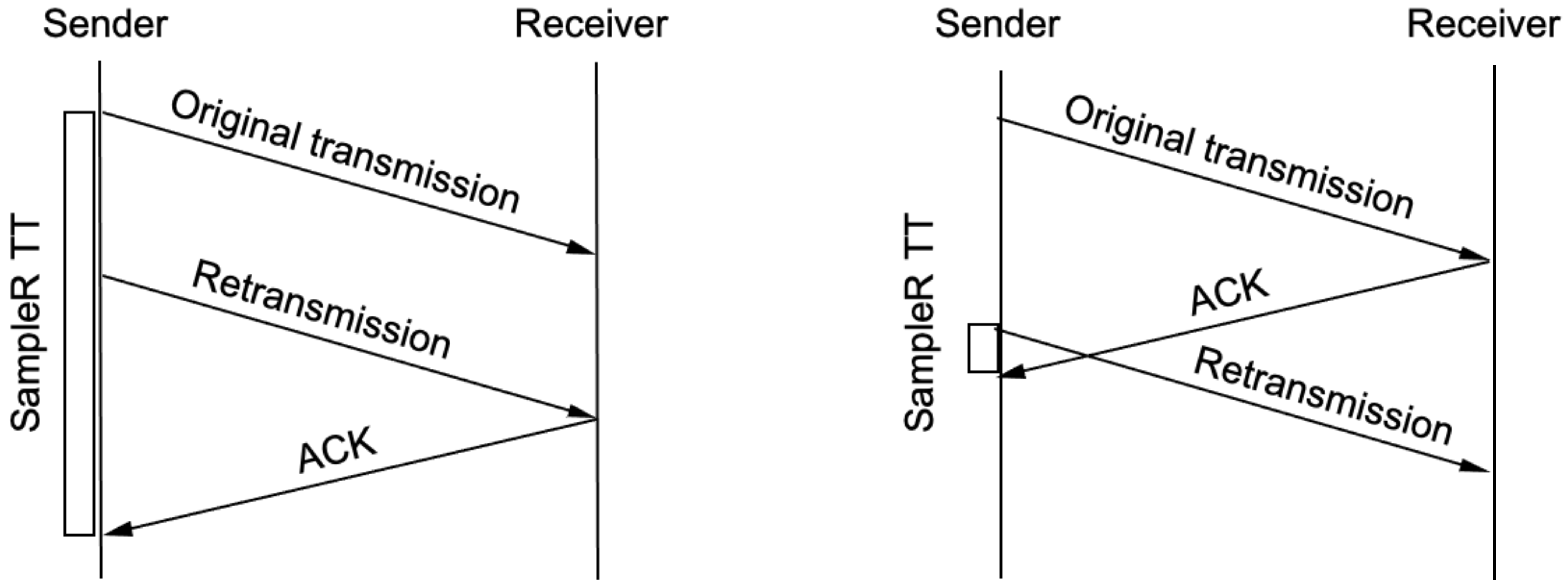
Solution: Nagle's Algorithm

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
```

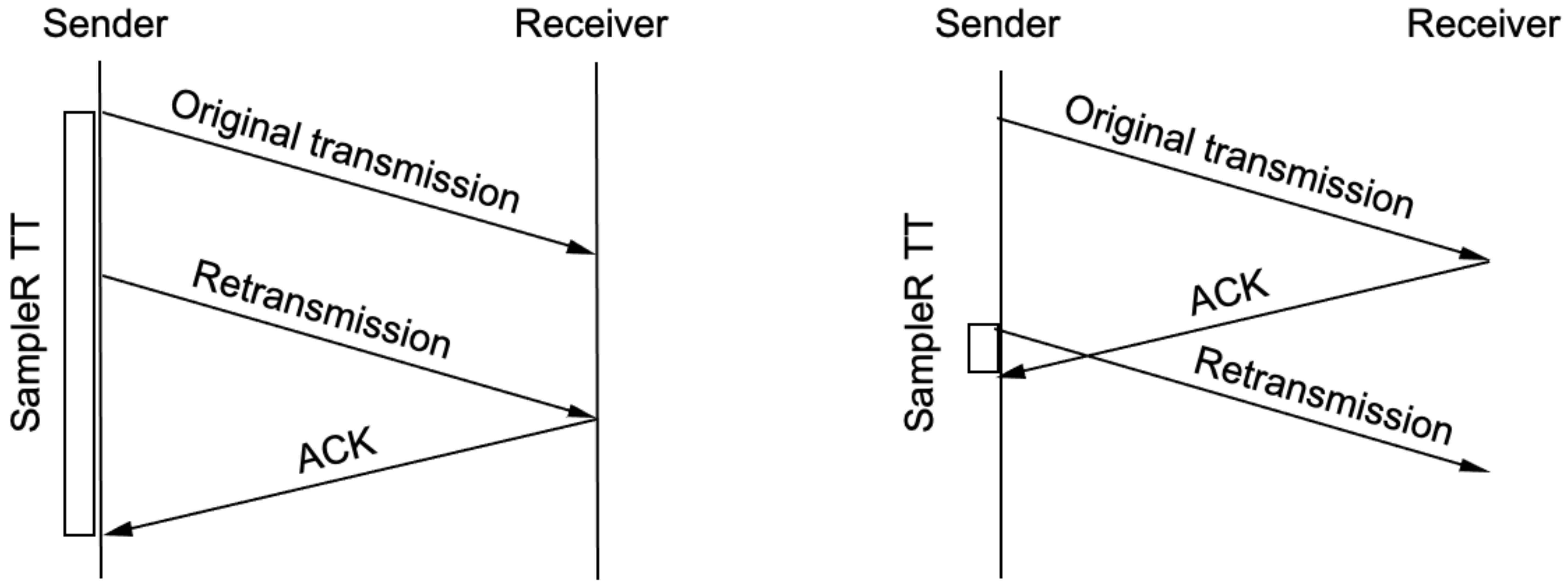
Issue #2: Timeout Setup during Retransmission



Two degenerate cases

- Do not sample RTT when retransmitting

Solution: Karn/Partridge Algorithm for RTO



After each retransmission, set the next RTO to be double the value of the last

- Exponentially backoff is a well-known control theory method
- Loss is most likely caused by congestion so be careful

Issue #3: Retransmitted Segments

What segments are retransmitted under a timeout?

- Option #1: retransmit all segments subsequently after the missing one (pessimistic)
- Option #2: retransmit just the missing one (optimistic)

Issue #3: Retransmitted Segments

What segments are retransmitted under a timeout?

- Option #1: retransmit all segments subsequently after the missing one (pessimistic)
- Option #2: retransmit just the missing one (optimistic)
- Option #3: 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: retransmit all segments subsequently after the missing one (pessimistic)
- Option #2: retransmit just the missing one (optimistic)
- Option #3: selective acknowledgment
  - The receiver uses optional fields to acknowledge the missing ones
  - SACK option

Tell the sender what segments have been arrived

Solution: TCP SACK

Selective Acknowledgements (SACK)

- **#1:** Same congestion control mechanisms as TCP RENO
  - Uses TCP options fields
  - Timeouts are still used
- **#2:** When out-of-order data arrives, tell the sender which segments have been received
  - Enables the sender to maintain an image of the receiver's queue
- **#3:** Sender then resends all missing segments without waiting to timeout
  - Doesn't send beyond CWND
  - When no old data needs to be resent, then send new data

Issue #4: TCP Reno is not the only approach

TCP Vegas: source watches for some sign that router's queue is building up and congestion will happen

- RTT grows
- Sending rate flattens

Solution: Host-centric Congestion Avoidance

#1: Vegas tries to control the sending rate to avoid buffers to be filled

#2: Let BaseRTT be the minimal of all measured RTTs

#3: If not overflowing the connection, then

- ExpectedRate = CongestionWindow/BaseRTT

#4: Source calculates sending rate (ActualRate) per RTT

- Pick one packet per RTT, timestamp send/ACK packet pair, and divide by the number of bytes in transit

Vegas Algorithm

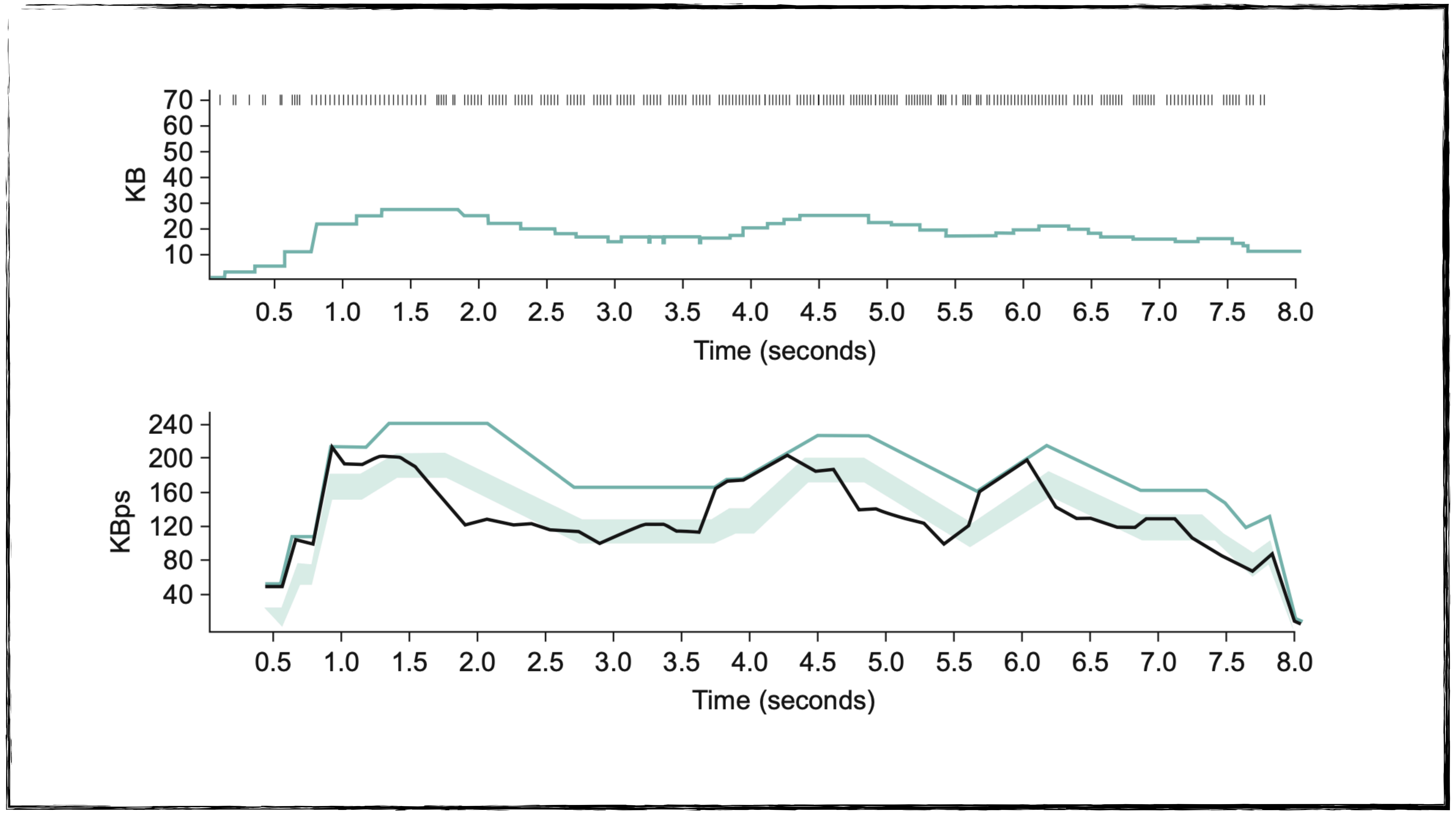
Source compares ActualRate with ExpectRate

- Diff = ExpectedRate - ActualRate
- If Diff < alpha
  - Increase CongestionWindow linearly
- Else if Diff > beta
  - Decrease CongestionWindow linearly
- Else
  - Leave CongestionWindow unchanged

Vegas Results

Trace

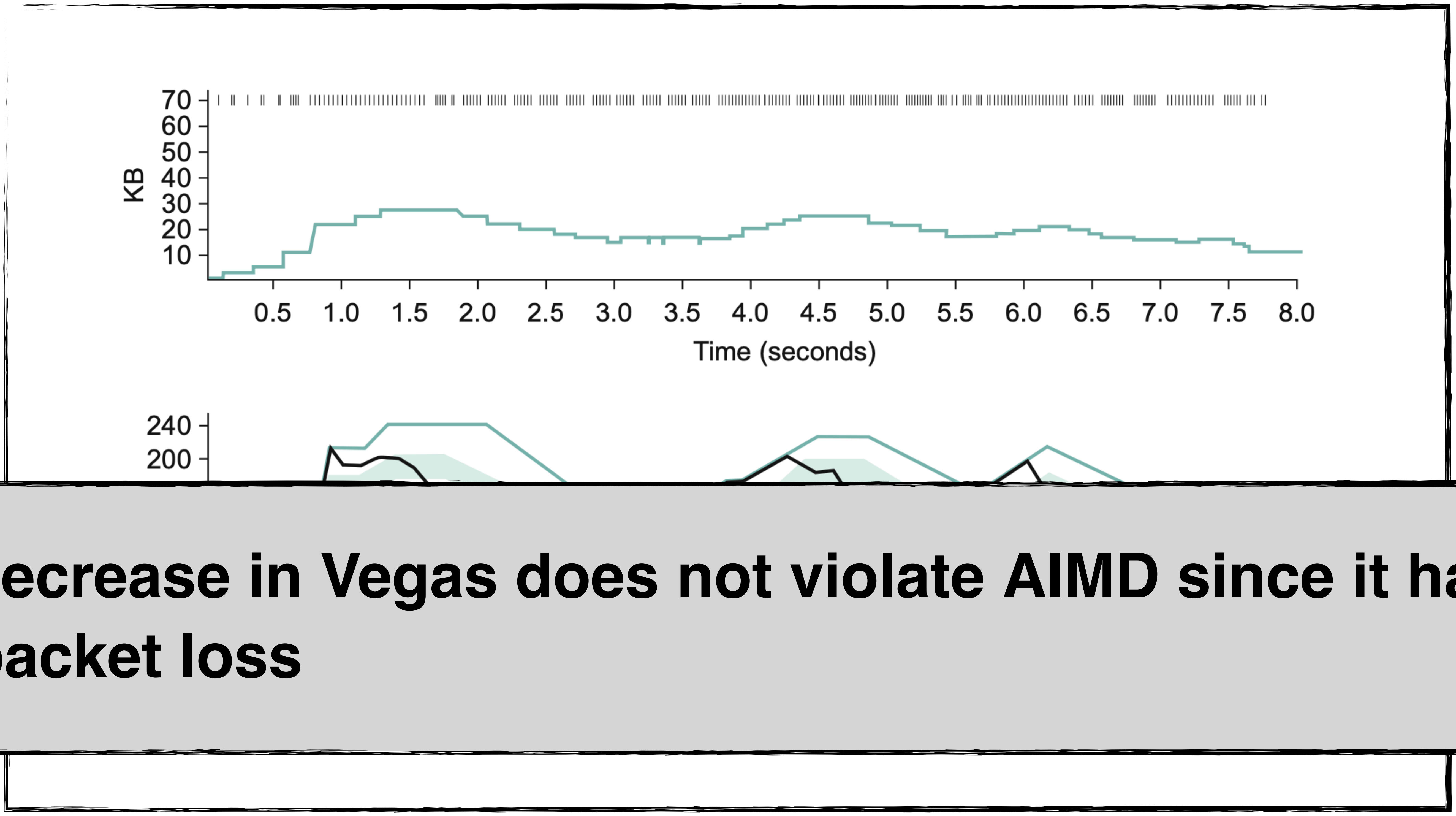
- alpha = 30KBps, beta = 60KBps



Vegas Results

Trace

- alpha = 30KBps, beta = 60KBps



Linear decrease in Vegas does not violate AIMD since it happens before packet loss

Terminology

1. Host
2. NIC
3. Multi-port I/O bridge
4. Protocol
5. RTT
6. Packet
7. Header
8. Payload
9. BDP
10. Baud rate
11. Frame/Framing
12. Parity bit
13. Checksum
14. Ethernet
15. MAC
16. (L2) Switch
17. Broadcast
18. Acknowledgement
19. Timeout
20. Datagram
21. TTL
22. MTU
23. Best effort
24. (L3) Router
25. Subnet mask
26. CIDR
27. Converge
28. Count-to-infinity
29. Line card
30. Network processor
31. Gateway
32. Private network
33. IPv6
34. Multicast
35. IGMP
36. SDN
37. (Transport) port
38. Pseudo header
39. SYN/ACK
40. Incarnation
41. Flow
42. SYN flood
43. TCP Segment
44. Window
45. Advertised Window
46. Effective Window
47. TCP Reno
48. Duplicated ACK
49. Congestion Window
50. Congestion Threshold
51. Selective Acknowledgment

Principle

1. Layering
2. Minimal States
3. Hierarchy
4. Mechanism/policy separation

Technique

1. NRZ Encoding
2. NRZI Encoding
3. Manchester Encoding
4. 4B/5B Encoding
5. Byte Stuffing
6. Byte Counting
7. Bit Stuffing
8. 2-D Parity
9. CRC
10. MAC Learning
11. Store-and-Forward
12. Cut-through
13. Spanning Tree
14. CSMA/CD
15. Stop-and-Wait
16. Sliding Window
17. Fragmentation and Reassembly
18. Path MTU discovery
19. DHCP
20. Subnetting
21. Supernetting
22. Longest prefix match
23. Distance vector routing (RIP)
24. Link state routing (OSPF)
25. Boarder gateway protocol (BGP)
26. Network address translation (NAT)
27. User Datagram Protocol (UDP)
28. Transmission Control Protocol (TCP)
29. Three-way Handshake
30. TCP state transition
31. EWMA
32. Sliding window
33. Flow control
34. AIMD
35. Slow start
36. Fast retransmit
37. Fast recovery
38. Nagle's algorithm
39. Karn/Partridge algorithm
40. TCP Vegas

Summary

Today's takeaways

#1: Nagle's algorithm improves the TCP efficiency by coalescing small segments

#2: The timeout threshold should be carefully configured when retransmission happens congestion control limits the number of outstanding bytes in the network

#3: Selective transmission improves the retransmission efficiency by deciding the specific missing segments

#4: TCP Vegas controls sending rate by ensuring no buffer overflow at the router

Next lecture

- TCP In-network Support