Introduction to Computer Networks

CS640

TCP Congestion Control (I)

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

Ming Liu
mgliu@cs.wisc.edu

Today

Last lecture

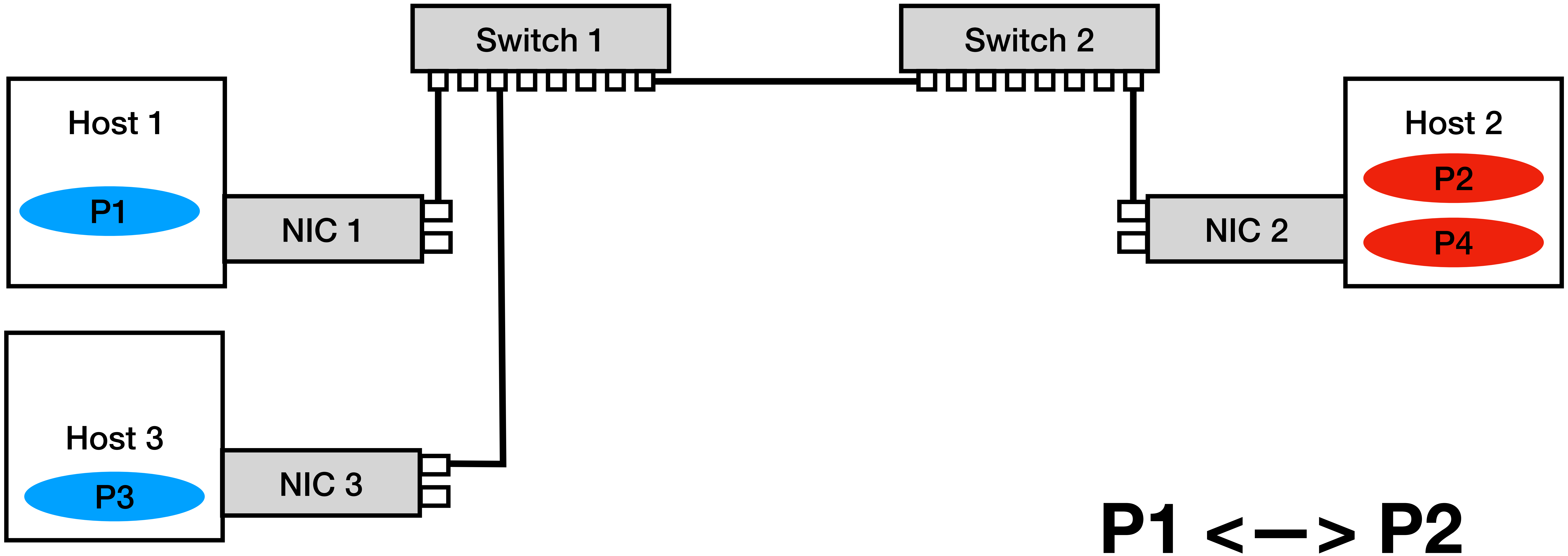
- How to ensure reliable data delivery?

Today

- How to share networking bandwidth among concurrent TCP flows?

Announcements

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



P1 <—> P2

P3 <—> P4

Q: What is the goal of TCP congestion control?

Q: What is the goal of TCP congestion control?

A: Utilization and fairness

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

Q: Why congestion control is hard?

Q: Why congestion control is hard?

A: Two challenges:

- #1: The available capacity of the underlying networking fabric keeps varying
- #2: Traffic is unpredictable

Q: What is the key idea behind congestion control?

Q: What is the key idea behind congestion control?

A: Window/Rate adjustment algorithm

- #1: Reaction point (RP) or sender } Adjust window/rate
- #2: Congestion point (CP) or switch/router
- #3: Notification point (NP) or receiver

} Issue implicit feedbacks (#2, #3)

Q: What is the key idea behind congestion control?

A: Window/Rate adjustment algorithm

- #1: Reaction point (RP) or sender } Adjust window/rate
- #2: Congestion point (CP) or switch/router
- #3: Notification point (NP) or receiver

} Issue implicit feedbacks

Mechanism and Policy

Q: How to adjust the window and issue implicit feedbacks?

Q: How to adjust the window and issue implicit feedbacks?

A: TCP Reno

- One of many congestion control algorithms
- Consists of three techniques

Technique #1: AIMD

Goal: adjust to changes in the available BW capacity

Technique #1: AIMD

Goal: adjust to changes in the available BW capacity

Additive Increase/Multiplicative Decrease

- Additive increase **CongestionWindow** when the congestion goes down
- Multiplicative decrease **CongestionWindow** when the congestion goes up

Technique #1: AIMD

Goal: adjust to changes in the available BW capacity

Additive Increase/Multiplicative Decrease

- Additive increase **CongestionWindow** when the congestion goes down
- Multiplicative decrease **CongestionWindow** when the congestion goes up

New state per flow: CongestionWindow

- Limits how much data source has in transit

MaxWin = MIN (Congestion Window, AdvertiseWindow)
EffWin = MaxWin - (LastByteSent - LastByteAcked)

Technique #1: AIMD

Goal: adjust to changes in the available BW capacity

Additive Increase/Multiplicative Decrease

- Additive increase **CongestionWindow** when the congestion goes down

(1). How to determine if the congestion goes up?

(2). How to determine if the congestion goes down?

(3). How much congestion window do we manipulate quantitatively?

Congestion goes up

A timeout occurs

- Packet loss
- Transmission slow

Congestion goes down

A CongestionWindow’s data are successfully delivered

- Each packet sent out during the last round-trip time (RTT) has been ACKed

CongestionWindow Manipulation

Increment CongestionWindow by one segment per RTT

- Congestion goes down
- AI phase: linear increase

Divide CongestionWindow by two if a timeout occurs

- Congestion goes up
- MD phase: multiplicative decrease – fast!!

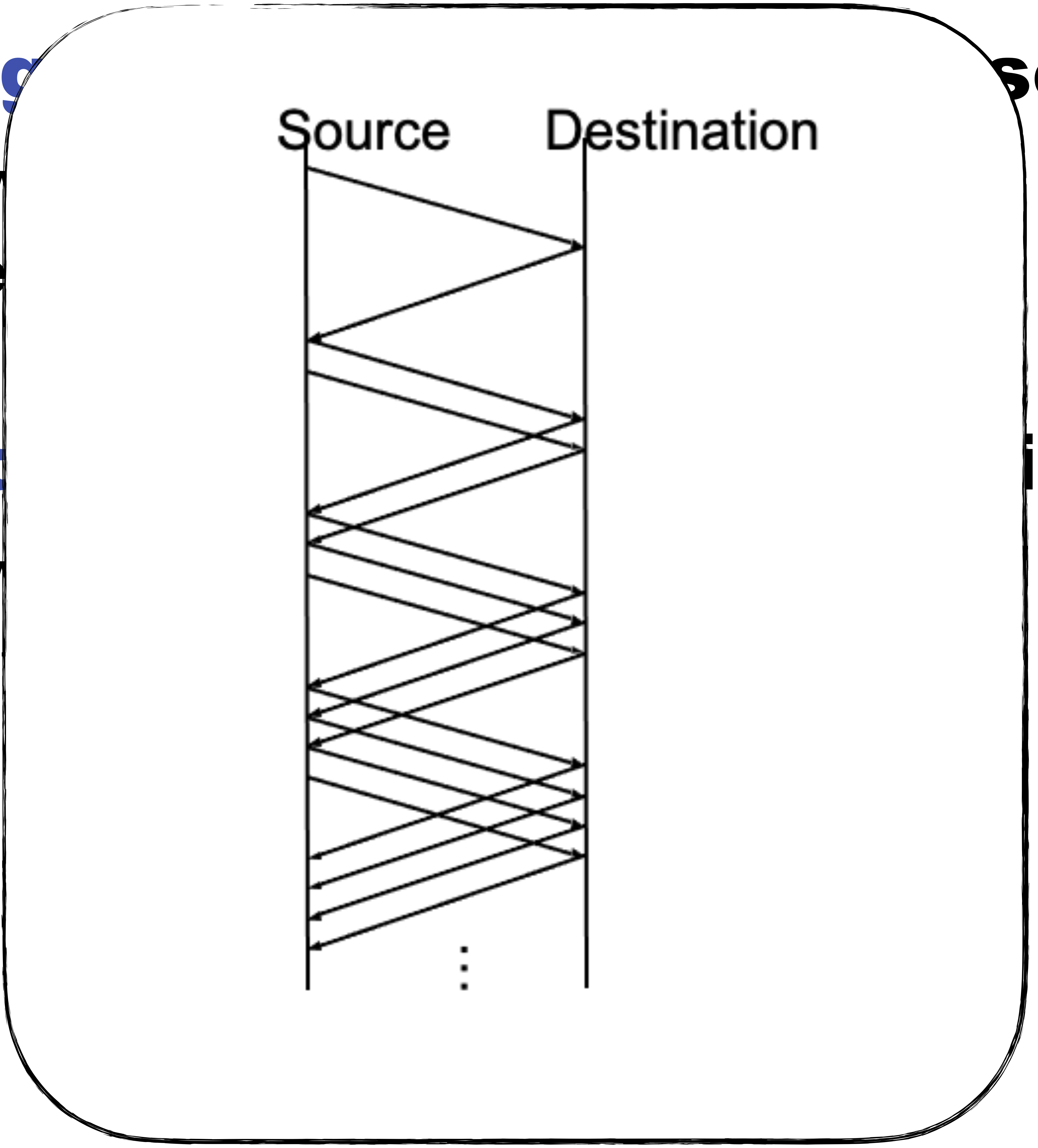
CongestionWindow Manipulation

Increment **Cong** segment per RTT

- Congestion goes dow
- AI phase: linear incre

Divide **Congest** imeout occurs

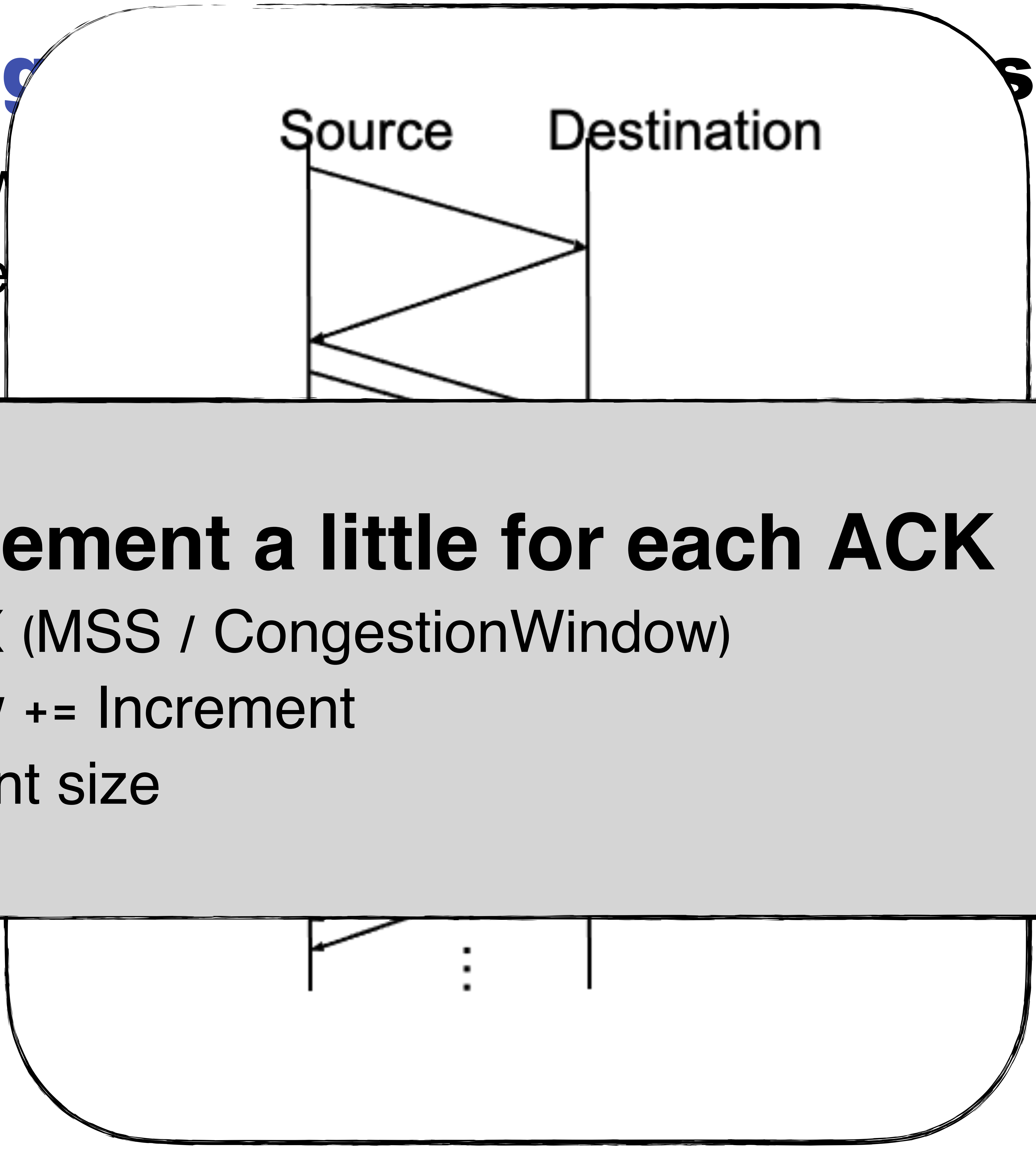
- Congestion goes dow
- MD phase: multiplica



CongestionWindow Manipulation

Increment Cong segment per RTT

- Congestion goes dov
- AI phase: linear incre

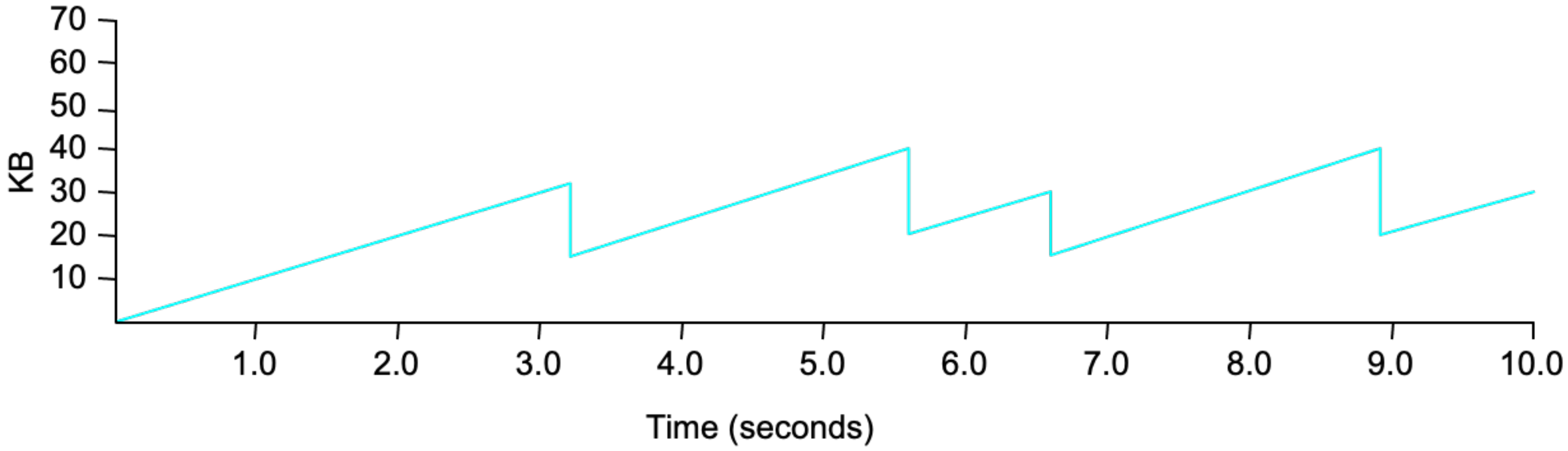


In practice, increment a little for each ACK

- Increment = MSS X (MSS / CongestionWindow)
- CongestionWindow += Increment
- MSS = max segment size

AIMD Simulation Result

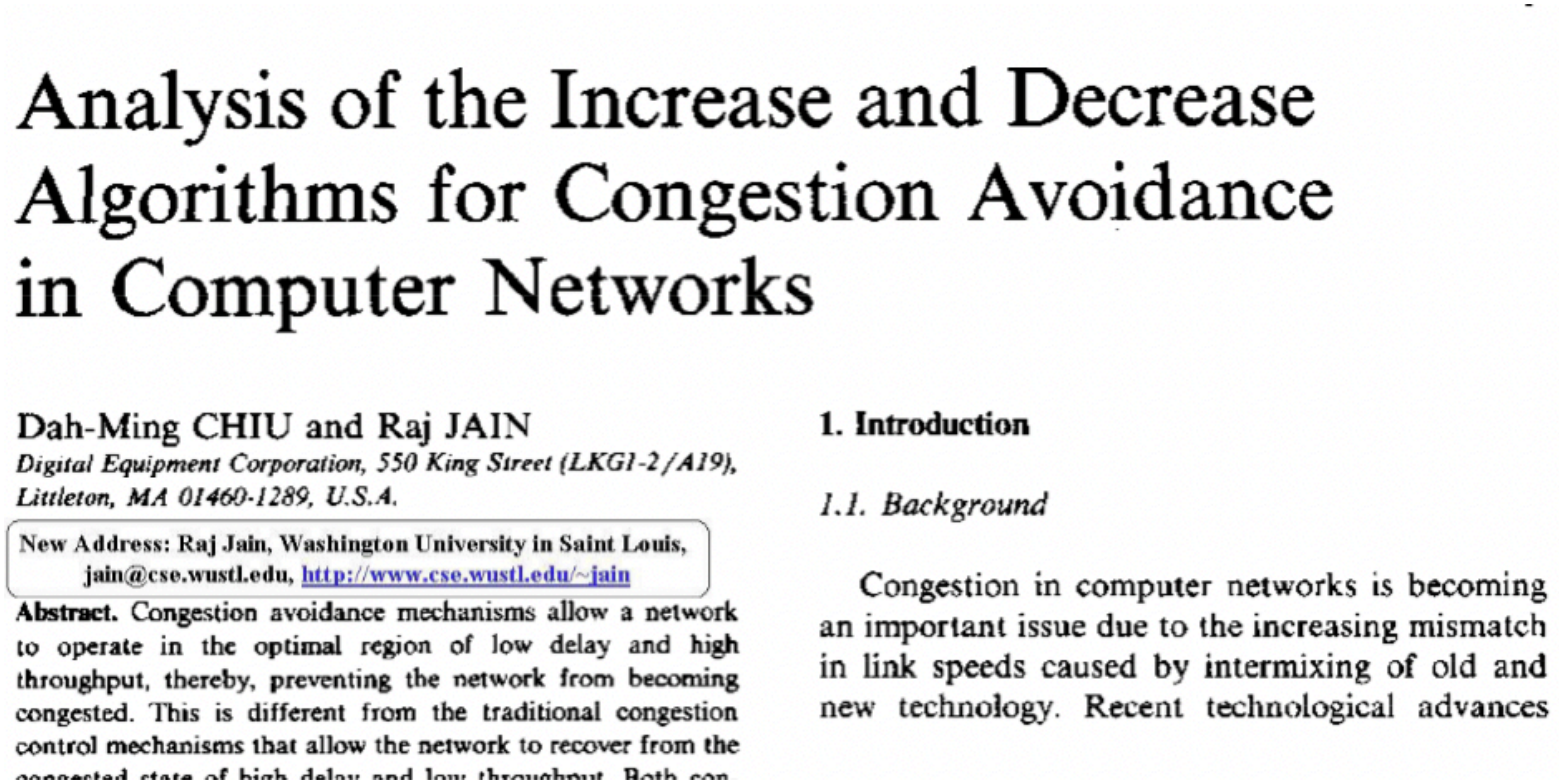
Trace: sawtooth behavior



AIMD Discussion

Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks, 1989

- Efficiency
- Fairness
- Convergence
- Distributedness



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

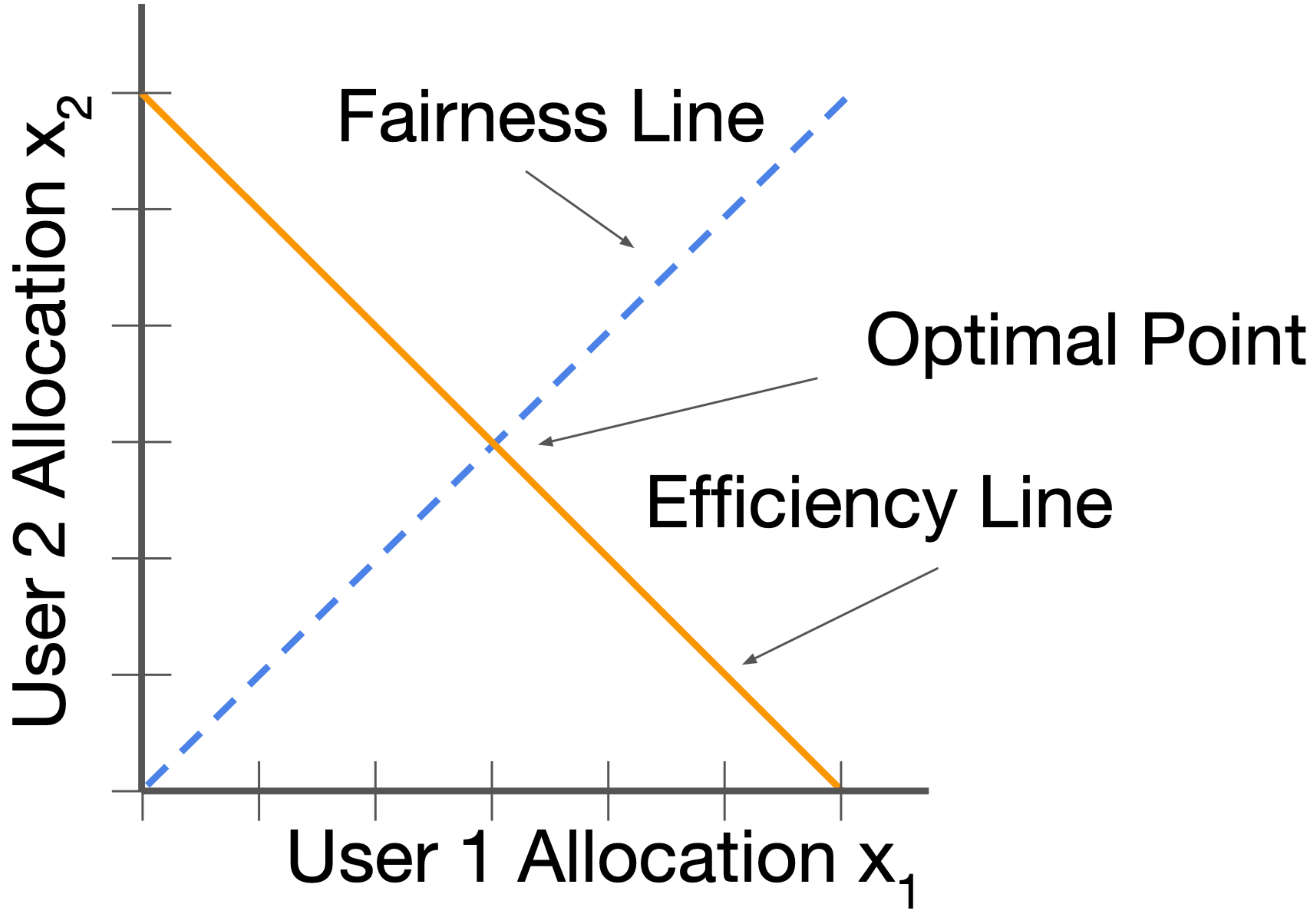
1. Introduction

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

Takeaway

A good congestion control mechanism should converge to the optimal point. And AIMD can!



Technique #2: Slow Start

Goal: determine the available capacity in the first

Option #1: Additive increase

- One segment per RTT
- Too slow

Option #2: Send as many segments as the advertised window allows

- Don't take the buffer space at routers into consideration
- No coordination with other flows

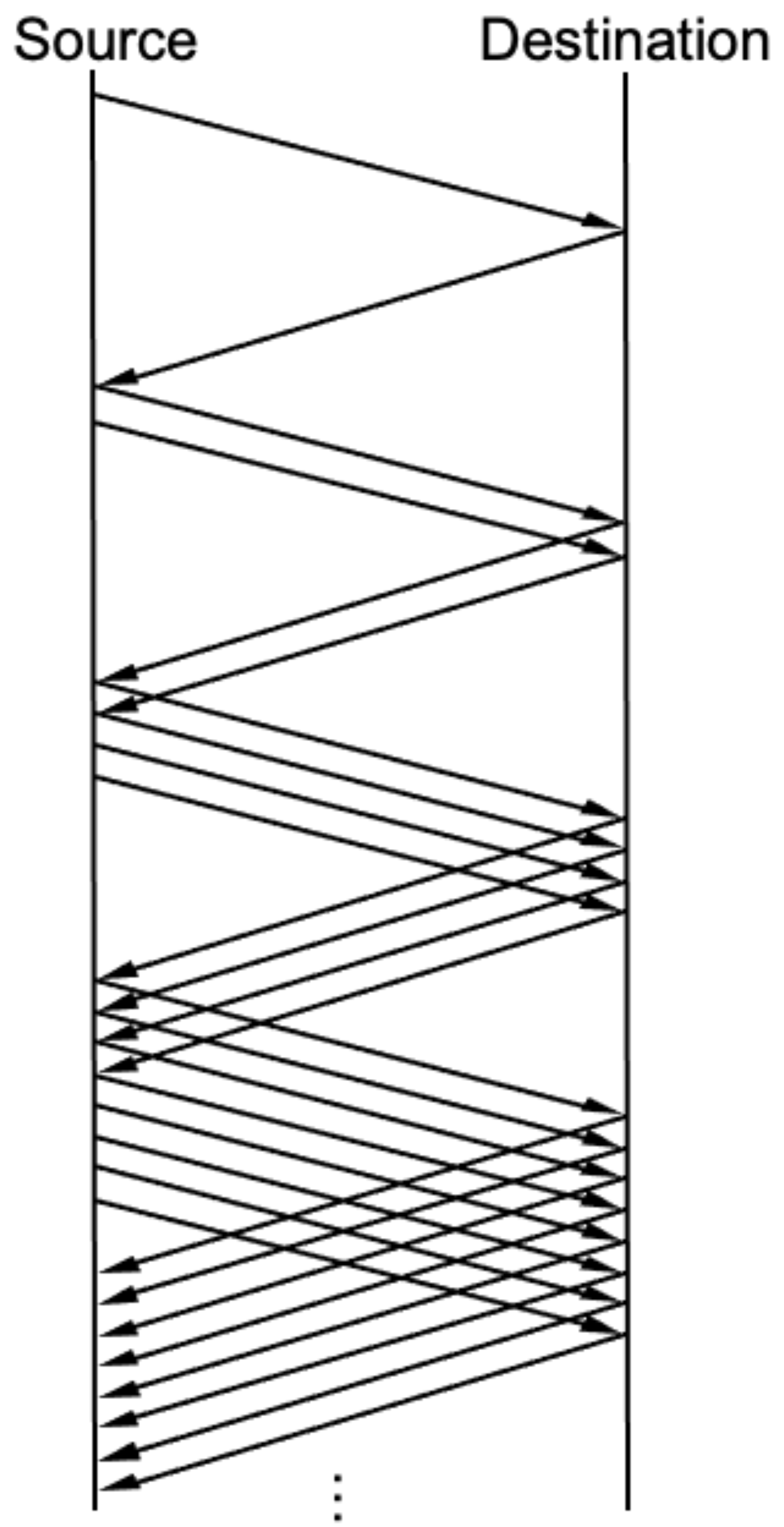
Technique #2: Slow Start

Goal: determine the available capacity in the first

Two steps:

- Step #1: being with CongestionWindow = 1 segment
- Step #2: double CongestionWindow each RTT

  => Increment by 1 segment for each ACK



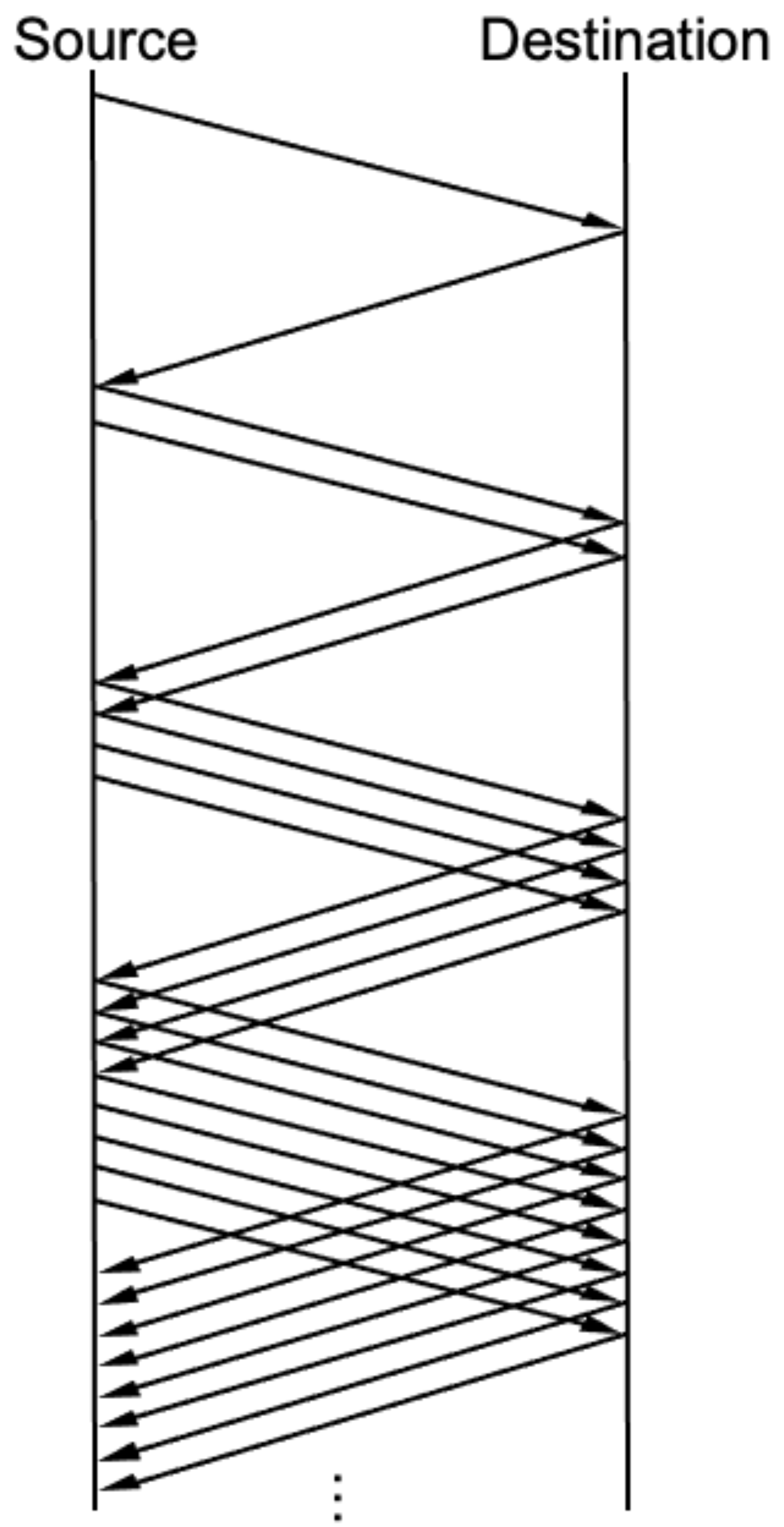
Technique #2: Slow Start

Goal: determine the available capacity in the first

Two steps:

- Step #1: being with CongestionWindow = 1 segment
- Step #2: double CongestionWindow each RTT

=> Increment by 1 segment for each ACK



Technique #2: Slow Start

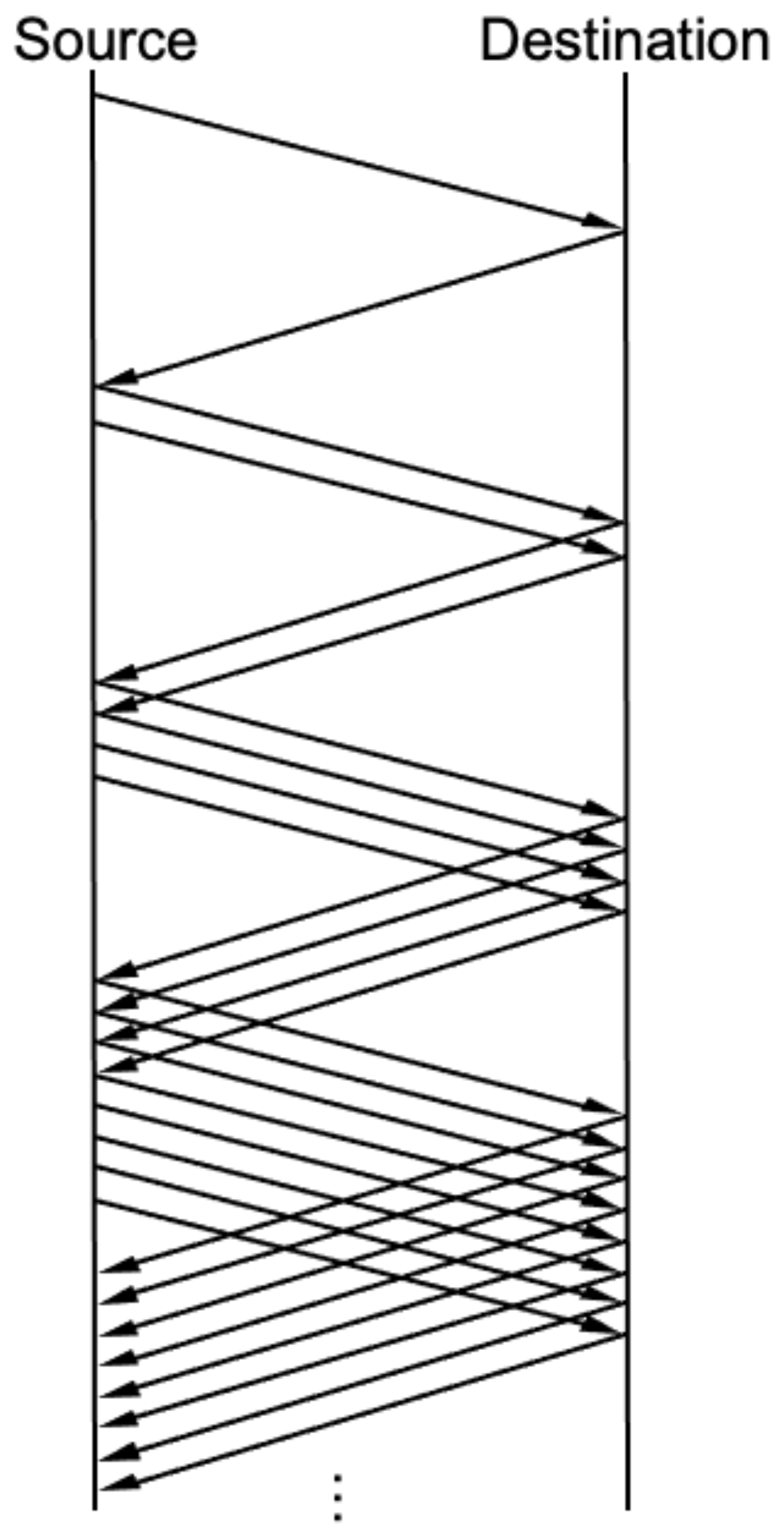
Goal: determine the available capacity in the first

Two steps:

- Step #1: being with CongestionWindow = 1 segment
- Step #2: double CongestionWindow each RTT

  => Increment by 1 segment for each ACK

Exponential increase to probe for available bandwidth



Slow Start Discussion

Used...

- A flow is just started
- When a connection goes dead waiting for timeout

A threshold (called CongestionThreshold) to decide when the slow start ends

- Also indicates when to begin additive increase

CongestionThreshold and CongestionWindow

#1: CongestionThreshold is typically set to a very large value on connection setup

#2: Set to CongestionWindow/2 on a time out

- So, CongestionThreshold goes through a multiplicative decrease for each packet loss
- Set CongestionWindow = 1
- CongestionThreshold and CongestionWindow always >= 1 MSS

#3: After the loss, when new data is ACKed, increase CongestionWindow

- If CongestionWindow <= CongestionThreshold, slow start
- Otherwise, additive increase

Technique #3-1: Fast Retransmit

Problem:

- Coarse-grained TCP timeouts lead to long idle periods

Technique #3-1: Fast Retransmit

Problem:

- Coarse-grained TCP timeouts lead to long idle periods

Solution:

- Add a heuristic that triggers the retransmission of a dropped packet sooner than the regular timeout mechanism

Duplicated ACK

NP or Receiver

- Resends the same acknowledgement when receiving the out-of-order segments

RP or Sender:

- Receives duplicated ACKs
- Packets might be lost or delayed

Duplicated ACK

NP or Receiver

- Resends the same acknowledgement when receiving the out-of-order segments

RP or Sender:

- Receives duplicated ACKs
- Packets might be lost or delayed

Fast retransmit: use 3 duplicate ACKs to trigger retransmission

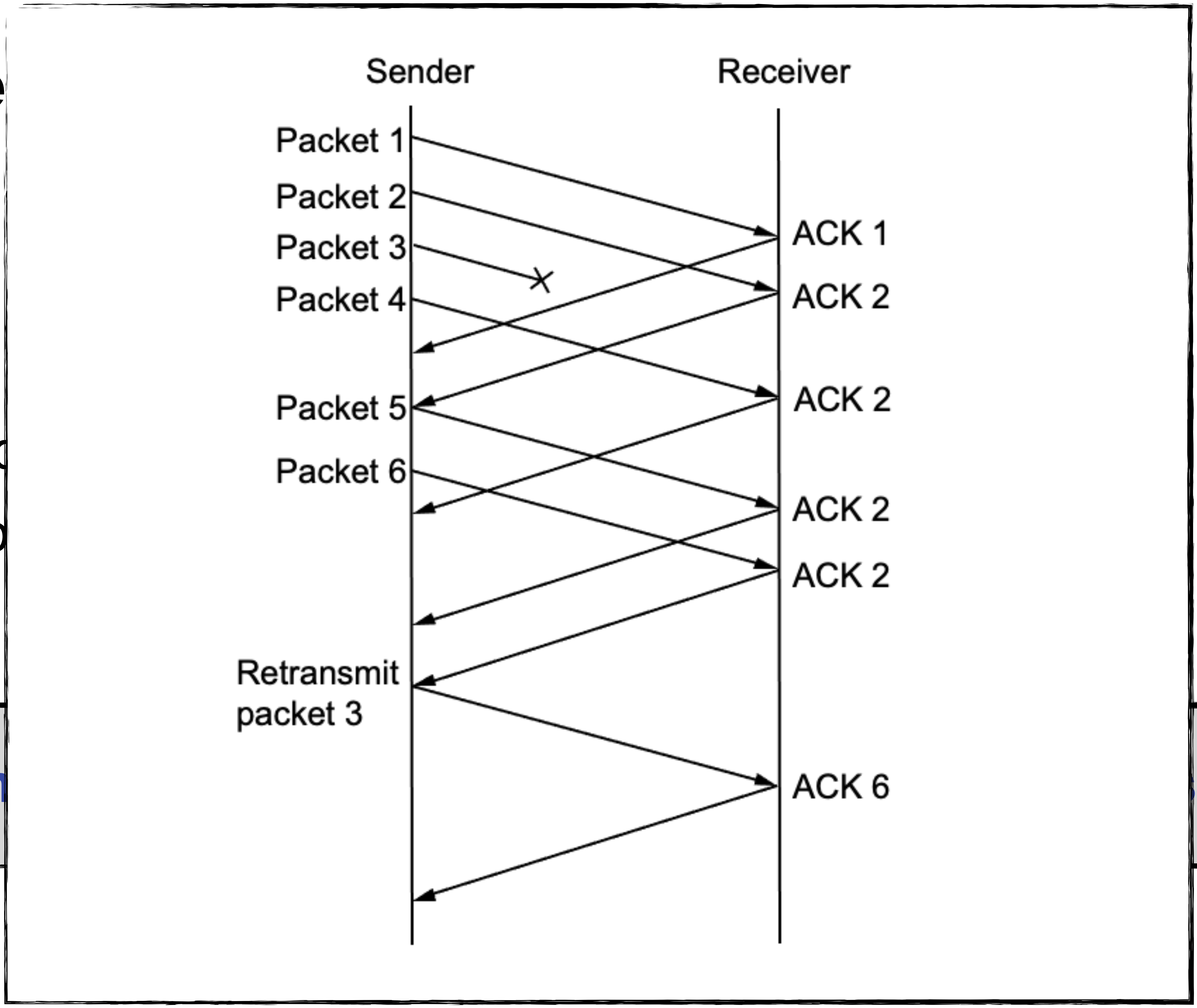
Duplicated ACK

NP or Receive

- Resends the same segments

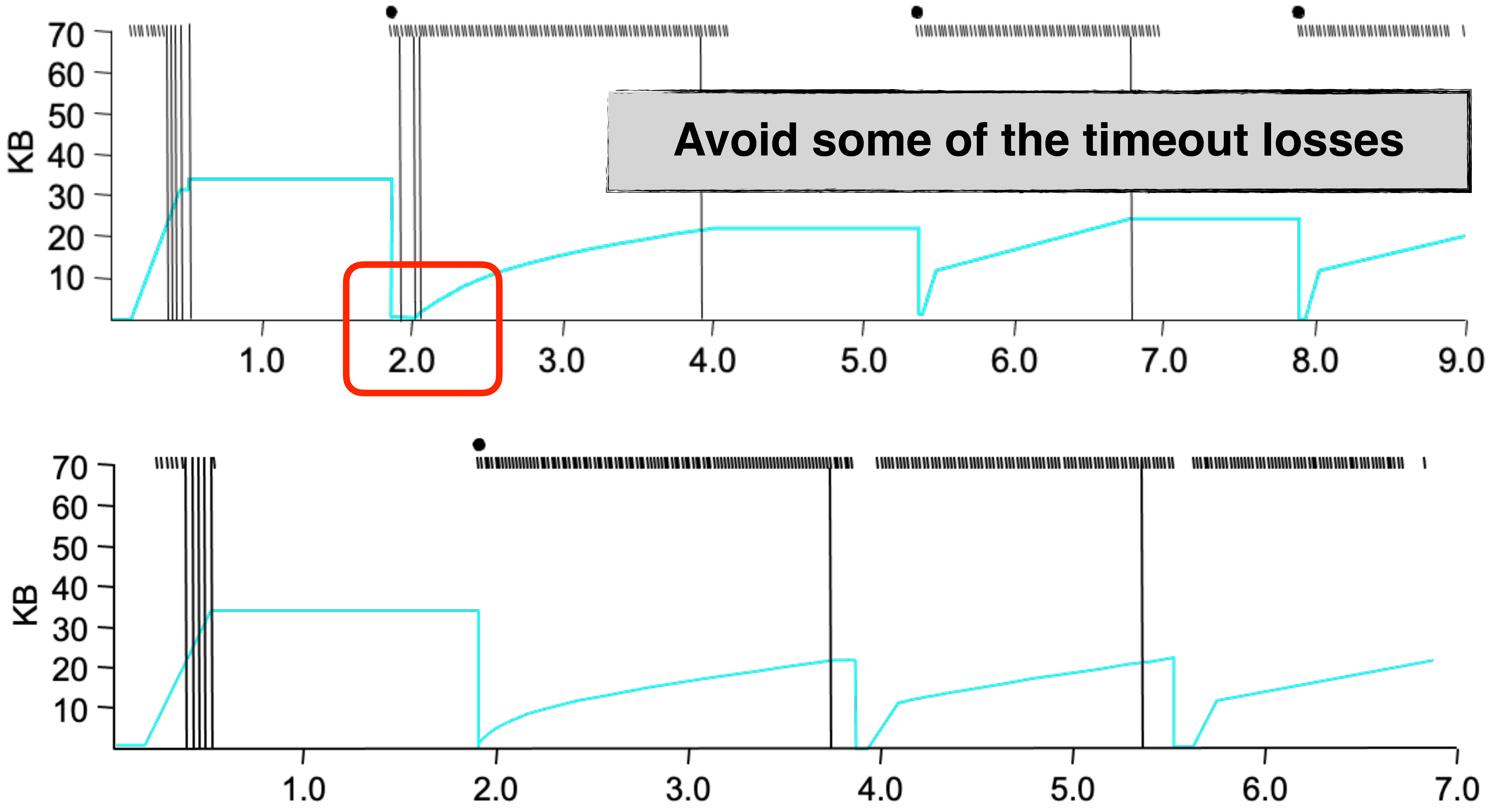
RP or Sender:

- Receives duplicated
- Packets might be lo



Fast retransm ion

Fast Retransmit Simulation Results



Technique #3-2: Fast Recovery

Problem:

- Slow start probing is unnecessary under duplicated ACKs

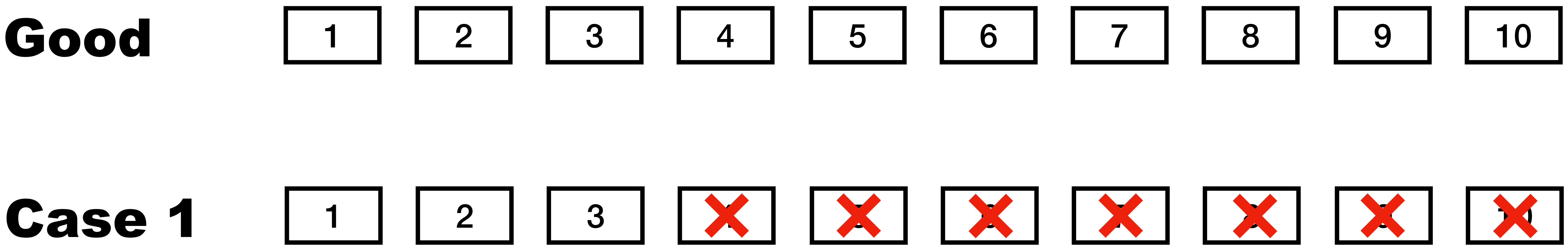
Solution:

- Adjust the **CongestionWindow** to half of the last successful **CongestionWindow**

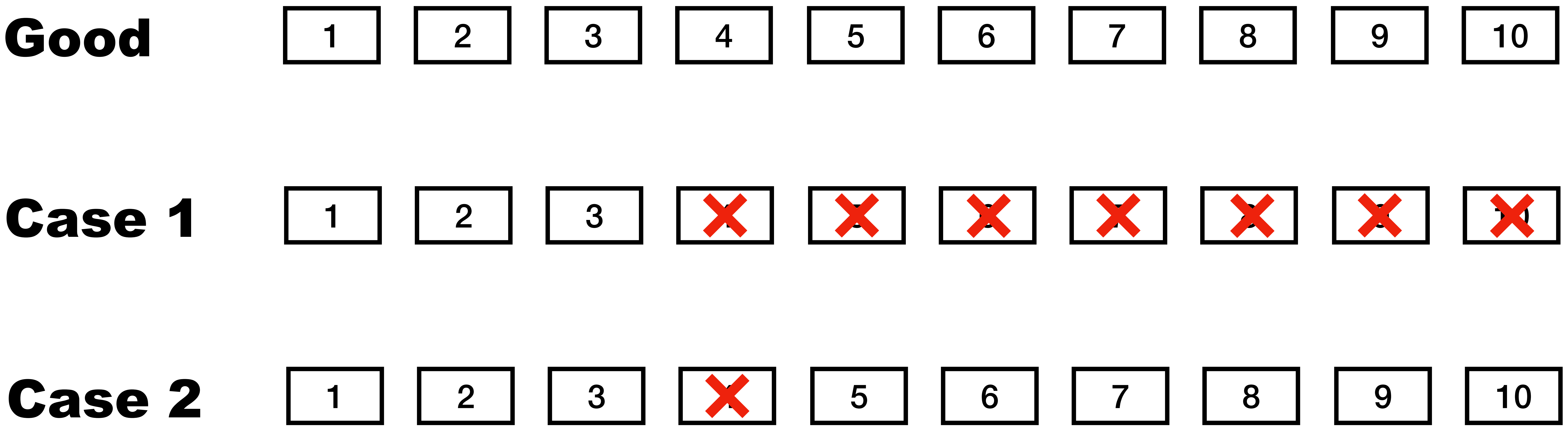
Dissecting Packet Loss

Good	1	2	3	4	5	6	7	8	9	10

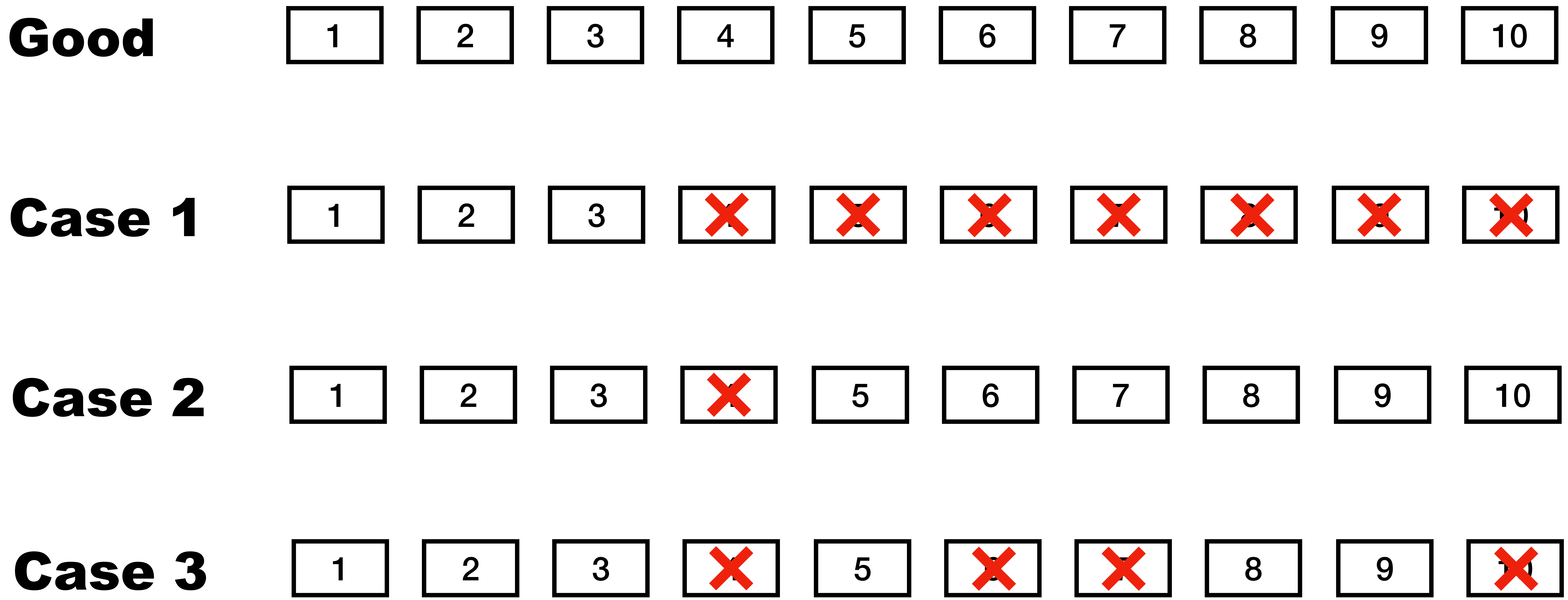
Dissecting Packet Loss



Dissecting Packet Loss



Dissecting Packet Loss



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

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

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

Summary

Today's takeaways

#1: TCP congestion control limits the number of outstanding bytes in the network

#2: TCP Reno consists of three techniques: AIMD, slow start, fast retransmit/recovery

Next lecture

- TCP congestion control (II)