Lesson 3.5: **Transport Layer**

CSC450 – COMPUTER NETWORKS | WINTER 2019-20

DR. ANDREY TIMOFEYEV

OUTLINE

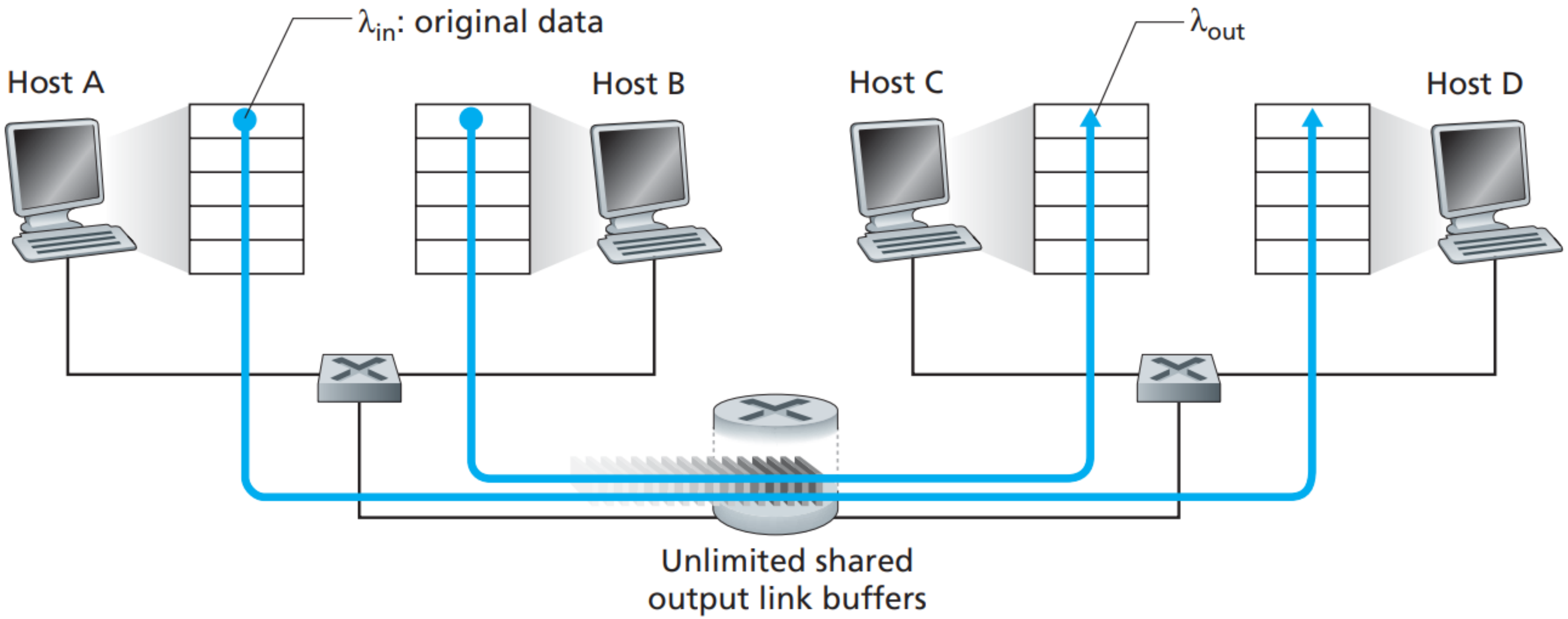
- Principles of congestion control.
- Approaches to congestion control.
- TCP congestion control.
  - Slow start.
  - Congestion avoidance.
  - Fast recovery.

PRINCIPLES OF CONGESTION CONTROL (1)

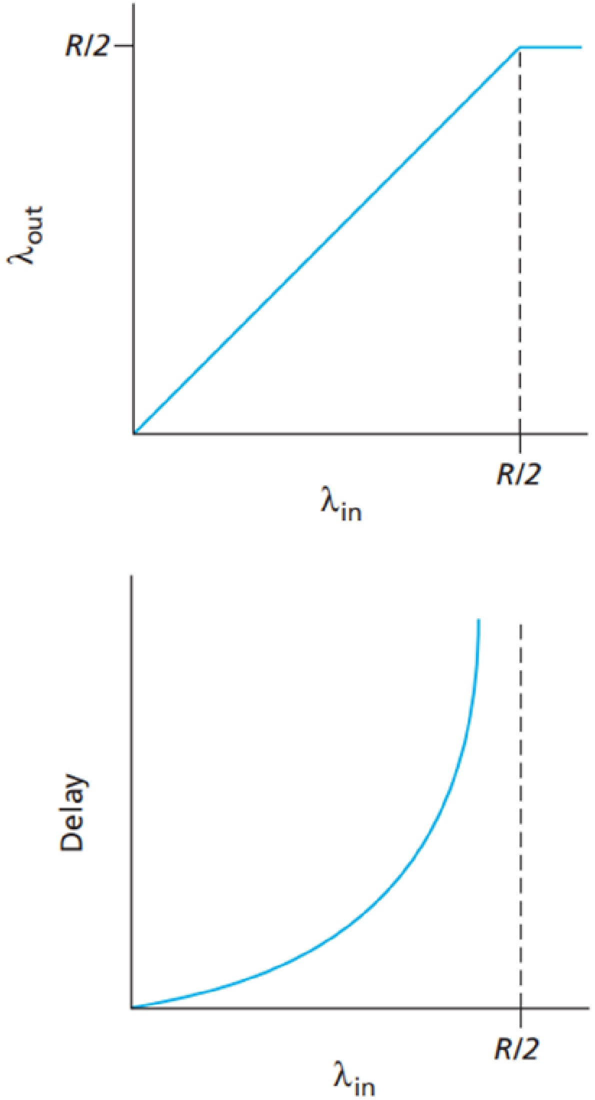
- **Network congestion** – **reduced** quality of **service** that occurs when a network node/link is carrying **more data** than it can **handle**.
  - *Informally* – **too many** sources sending **too much** data **too fast** for the network to handle.
- Network congestion **effects**:
  - **Lost packets** (router buffer overflow).
  - **Long delays** (router buffer queueing).
- **Packet retransmission** only treats the *symptom*, but not the *cause* of the **network congestion**.
- **Solution** – mechanism that **throttles** senders **rate** once congestion is **perceived**.

PRINCIPLES OF CONGESTION CONTROL (2)

- **Scenario 1:** Two senders & router with infinite buffer.



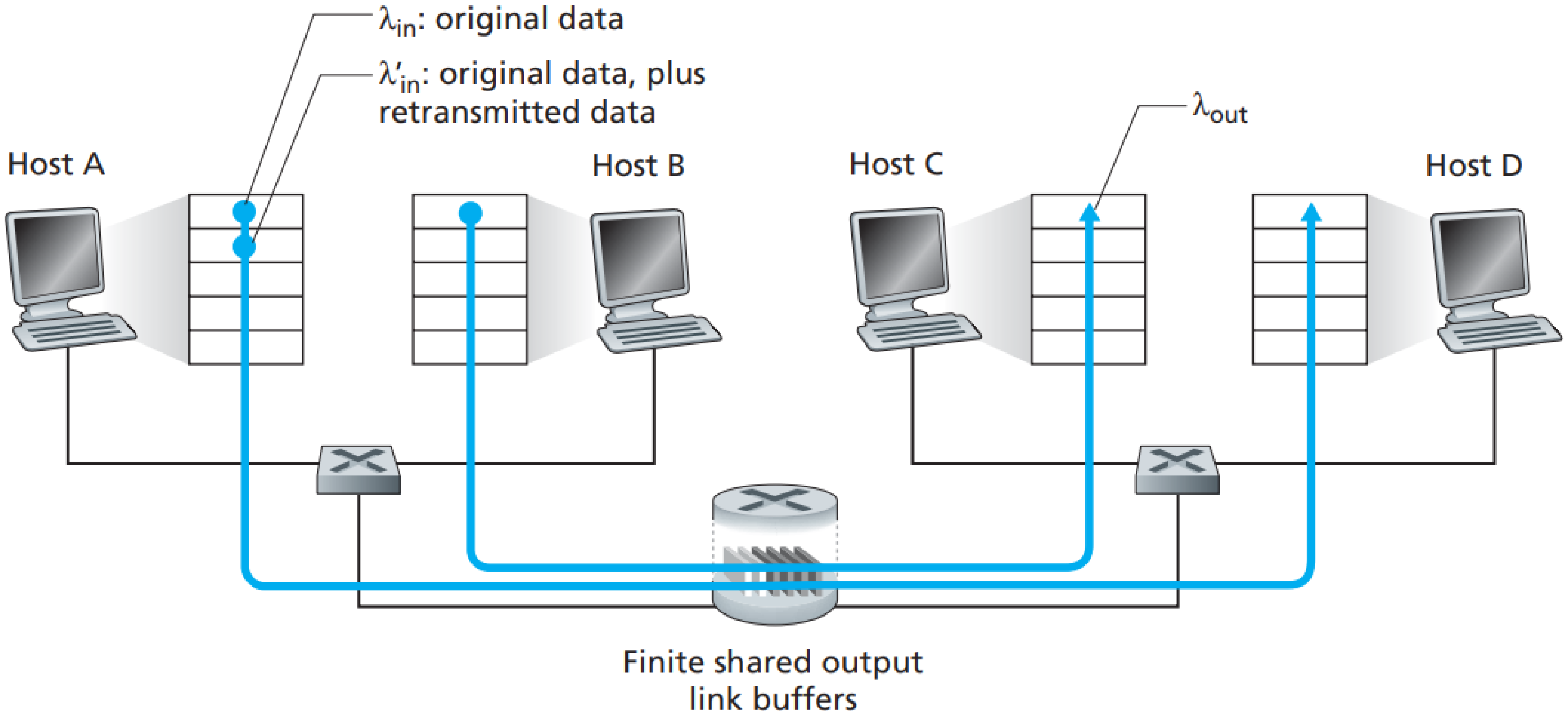
Two hosts share a single router with infinite buffer



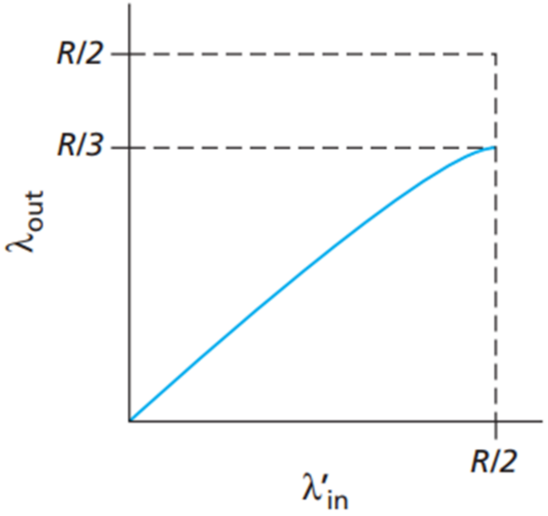
Throughput and delay

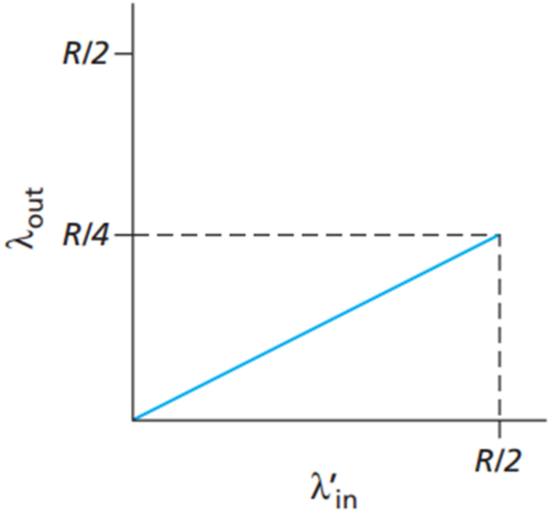
PRINCIPLES OF CONGESTION CONTROL (3)

- **Scenario 2:** Two senders & router with finite buffer.



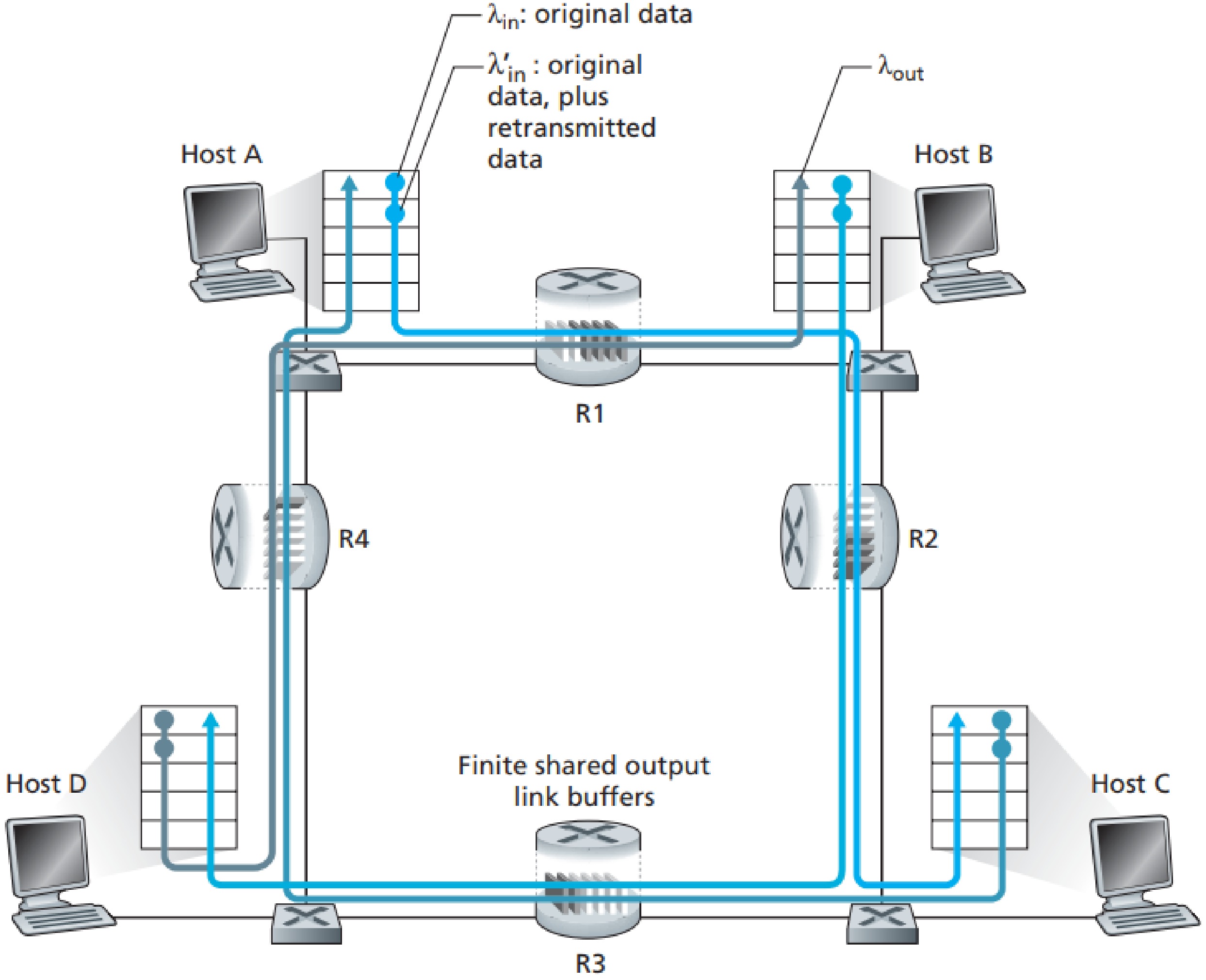
Two hosts (with retransmissions) and a router with finite buffer

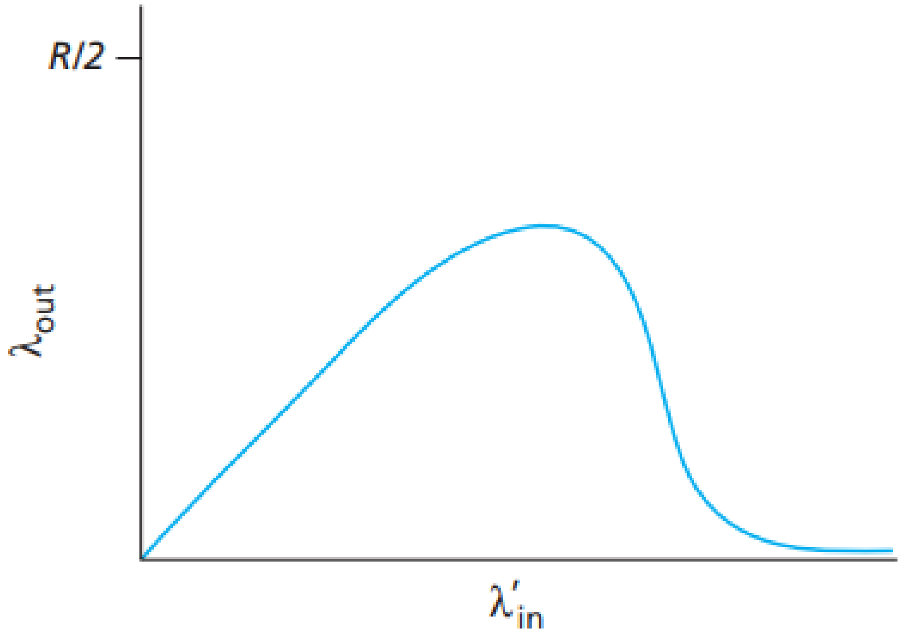




PRINCIPLES OF CONGESTION CONTROL (4)

- **Scenario 3**: Four senders, routers with finite buffers & multi-hop paths.





Throughput with finite buffers and multi-hop paths

APPROACHES TO CONGESTION CONTROL

- Two **approaches** to **control congestion** in networks:
  - **End-to-end congestion control**.
    - No support from network layer.
    - Congestion is inferred by transport layer based on packets loss and delays.
    - **TCP** congestion control approach.
  - **Network-assisted congestion control**.
    - Routers provide feedback to sender regarding congestion state of network.
      - Inform senders explicitly of transmission rate on outgoing link.
      - Provide congestion feedback in packet header.

TCP CONGESTION CONTROL: INTRO (1)

- **TCP** uses **end-to-end congestion** control.
  - *Reason*: IP (network layer) protocol does not provide explicit feedback regarding congestion.
- **TCP approach** for **congestion control**:
  - Each sender **limits** traffic **rate** sent into connection as a function of perceived network **congestion**.
    - Perceives **congestion** $\rightarrow$ **reduces** sending rate.
    - Perceives **no congestion** $\rightarrow$ **increases** sending rate.

TCP CONGESTION CONTROL: INTRO (2)

- **How does TCP sender limit the rate at which it sends traffic into connection?**
  - **Sender** side keeps track of additional **variable** – **congestion window** (*cwnd*).
    - Constraint on the senders' traffic sending rate.
  - *LastByteSent – LastByteAcked ≤ min{cwnd, rwnd}*
  - **TCP sending rate** ≈ *cwnd / RTT* bytes/sec
    - Send *cwnd* bytes of data -> wait RTT for ACKs -> send more bytes.
    - By **adjusting** *cwnd*, the sender adjusts the **rate** at which it **sends** data into connection.

TCP CONGESTION CONTROL: INTRO (3)

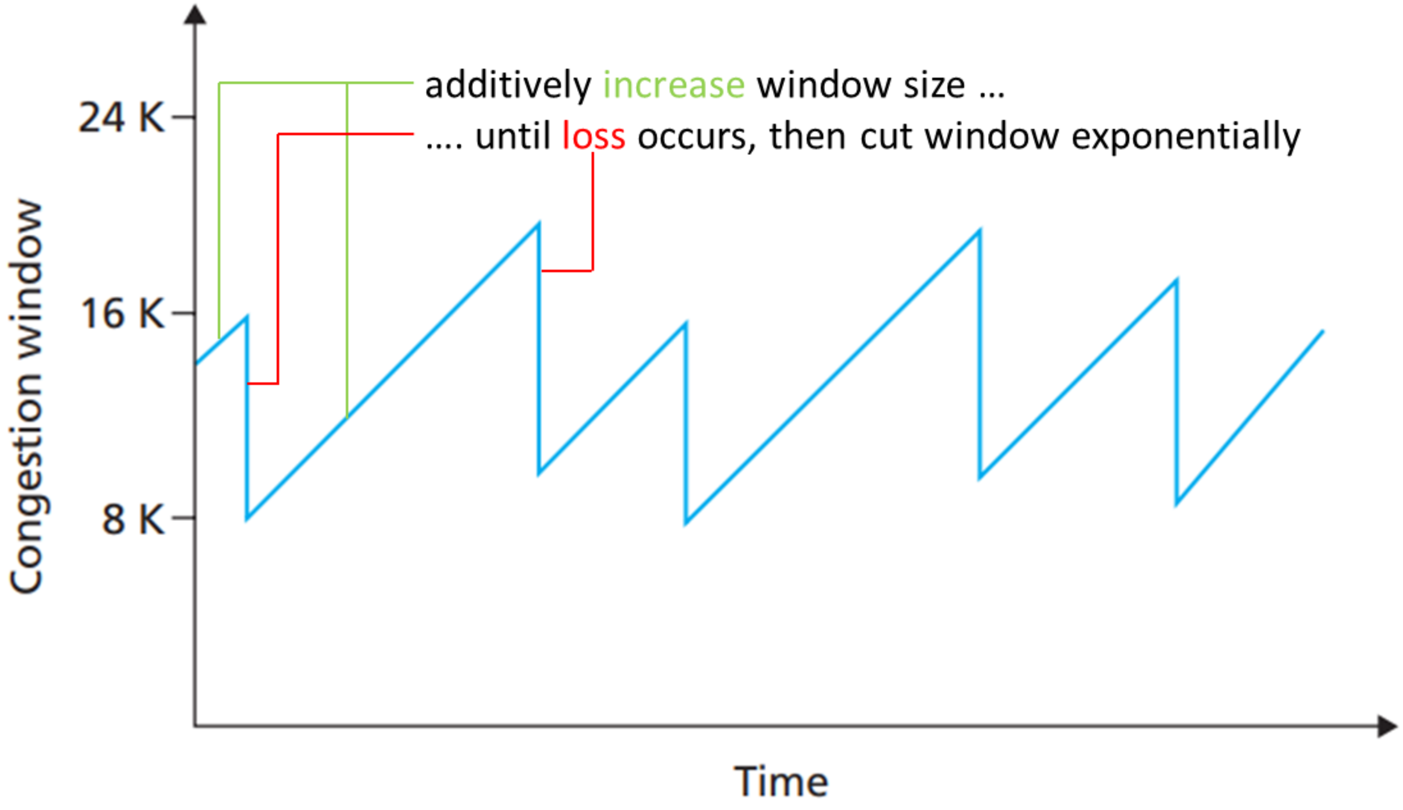
- **How does TCP sender perceives there is a congestion on the path to the destination?**
  - Sender **timeouts** or **receives** three duplicated **ACKs** – network is **congested**.
    - Congestion **window decreased** $\rightarrow$ sending **rate decreased**.
  - Sender **receives** (non-duplicated) **ACKs** – network is **not congested**.
    - Congestion **window increased** $\rightarrow$ sending **rate increased**.

TCP CONGESTION CONTROL: INTRO (4)

- **How does TCP sender determine the rate at which it should send?**
  - Sending too **fast** – **congesting** network.
  - Sending too **slow** – **under-utilizing** network bandwidth.
  - TCP uses “**bandwidth probing**” to adjust sending rate:
    - **ACKed** segment $\rightarrow$ keep **increasing** sending rate until **lost** segment $\rightarrow$ **decrease** sending rate.
- **TCP congestion control** algorithm components:
  - Slow start.
  - Congestion avoidance.
  - Fast recovery (TCP Reno).

TCP CONGESTION CONTROL: AIMD

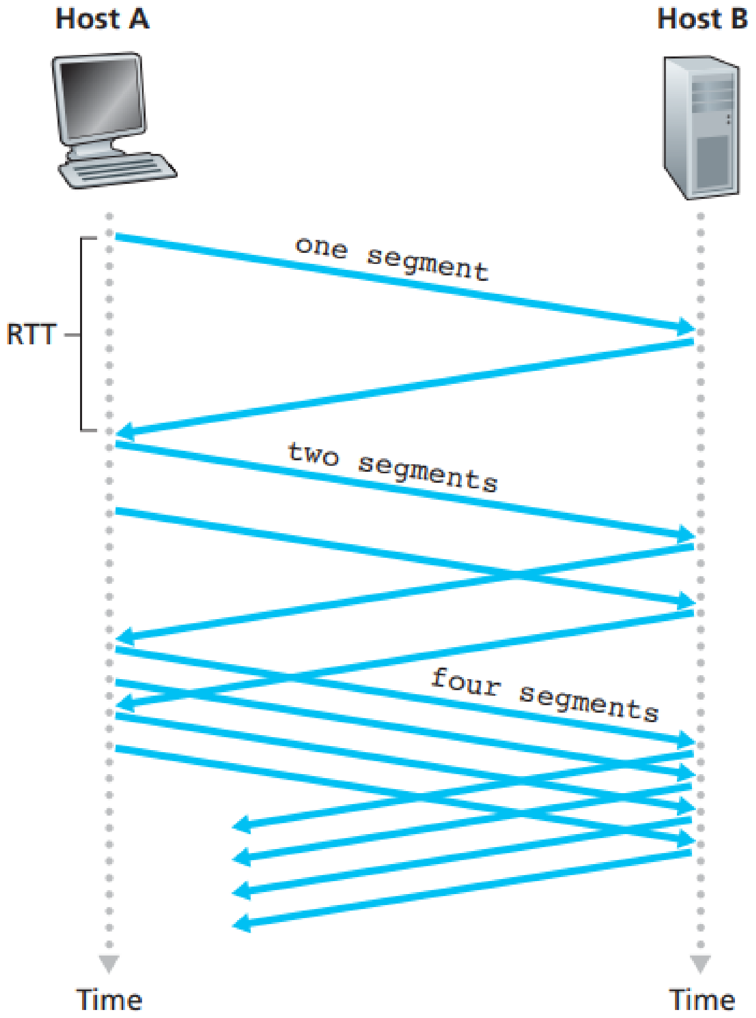
- **TCP congestion control** is characterized as **additive-increase, multiplicative-decrease** (**AIMD**).
  - *cwnd* increased **linearly** → **loss event** → *cwnd* decreased **exponentially**.



Additive-increase, multiplicative-decrease congestion control

TCP CONGESTION CONTROL: SLOW START

- **TCP congestion control slow start state:**
  - Event – **TCP connection begins**.
    - Set *cwnd = 1 MSS* (max segment size).
    - Initial sending rate = *MSS/RTT*.
  - Event – **ACK received**.
    - **Double** sending rate.
  - Event – **timeout**.
    - Set **slow start threshold** *ssthresh = cwnd / 2*.
    - Reset *cwnd = 1 MSS*.
    - Restart **slow start**.
  - Event – **three duplicated ACKs**.
    - Set *ssthresh = cwnd / 2*.
    - Reset *cwnd = ssthresh + 3 MSS*.
    - Enter **fast recovery** state.
  - Event – *cwnd ≥ ssthresh.*
    - Enter **congestion avoidance** state.



TCP slow start

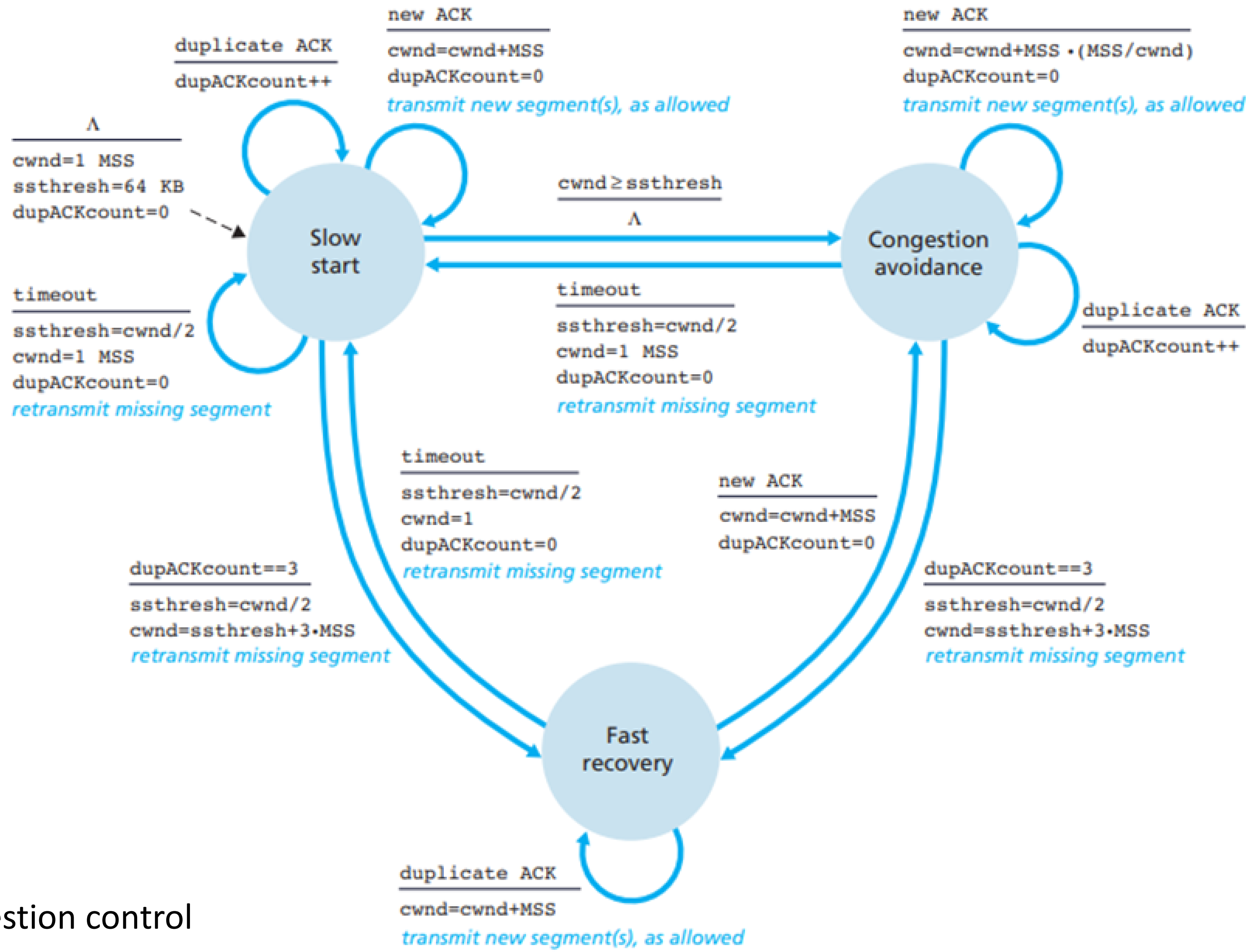
TCP CONGESTION CONTROL: CONGESTION AVOIDANCE

- **TCP congestion control congestion avoidance state:**
  - Event – **TCP enters congestion avoidance state**.
    - Initial *cwnd ≈ cwnd / 2*.
  - Event – **ACK received**.
    - Increase *cwnd = cwnd + 1 MSS*.
  - Event – **timeout**.
    - Reset *ssthresh = cwnd / 2*.
    - Reset *cwnd = 1 MSS*.
    - Enter **slow start** state.
  - Event – **three duplicated ACKs**.
    - Reset *ssthresh = cwnd / 2*.
    - Reset *cwnd = ssthresh + 3 MSS.*
    - Enter **fast recovery** state.

TCP CONGESTION CONTROL: FAST RECOVERY

- **TCP congestion control fast recovery state:**
  - Event – **TCP enters fast recovery state**.
    - Initial *cwnd ≈ ssthresh + 3 MSS*.
  - Event – **duplicated ACK received**.
    - Increase *cwnd = cwnd + 1 MSS*.
  - Event – **missing segment ACK received**.
    - *cwnd = cwnd + 1 MSS*.
    - Enter **congestion avoidance** state.
  - Event – **timeout**.
    - Reset *ssthresh = cwnd / 2*.
    - Reset *cwnd = 1 MSS*.
    - Enter **slow start** state.

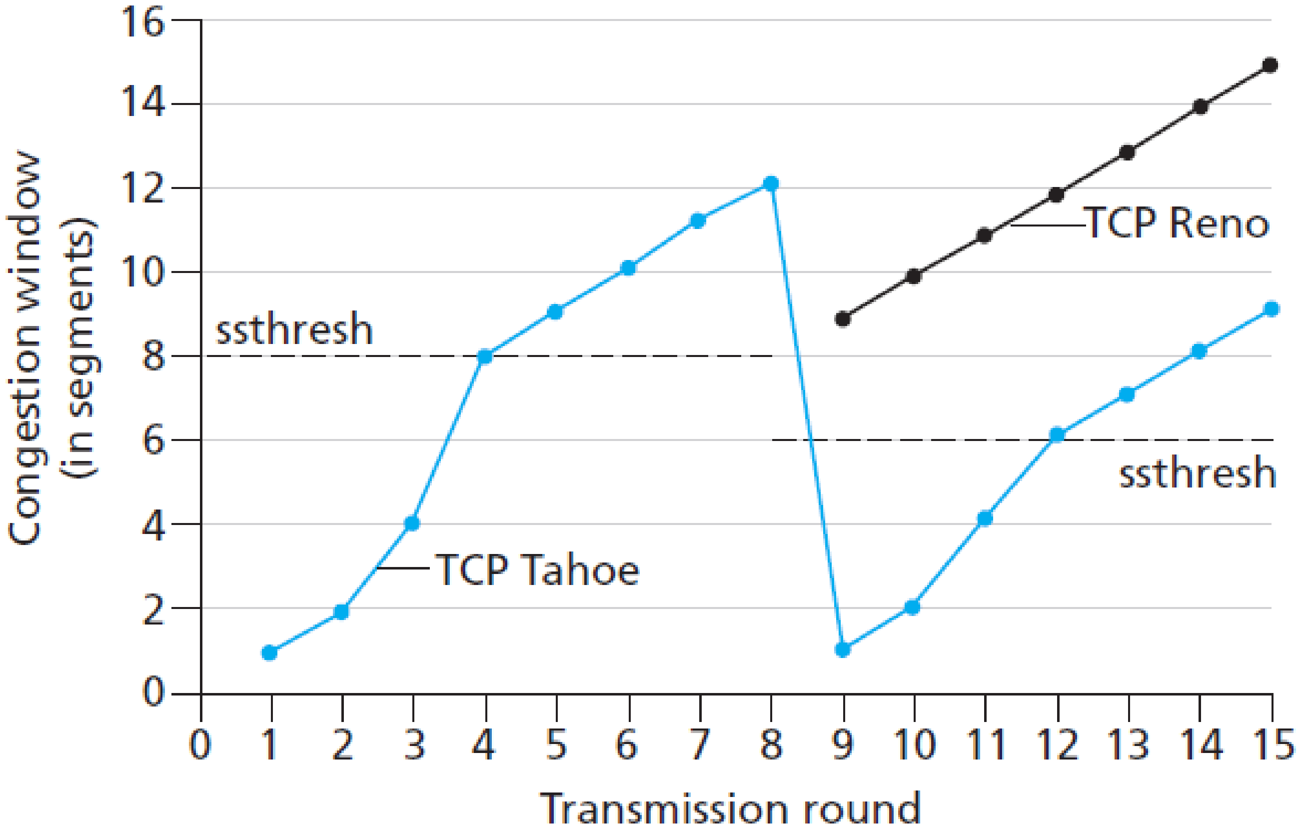
TCP CONGESTION CONTROL: STATES SUMMARY



States of TCP congestion control

TCP CONGESTION CONTROL: TAHOE VS RENO

- Two **types** of **TCP congestion control** implementation:
  - **TCP Tahoe** (no fast recovery state):
    - **Timeout** or **three duplicated ACKs**.
      - Reset *ssthresh = cwnd / 2*.
      - Reset *cwnd = 1 MSS*.
      - Enter **slow start** mode.
  - **TCP Reno:**
    - **Three duplicated ACKs:**
      - Enter **fast recovery** state.
    - **Timeout**:
      - Reset *ssthresh = cwnd / 2*.
      - Set *cwnd = ssthresh*.
      - Enter **congestion avoidance** state.

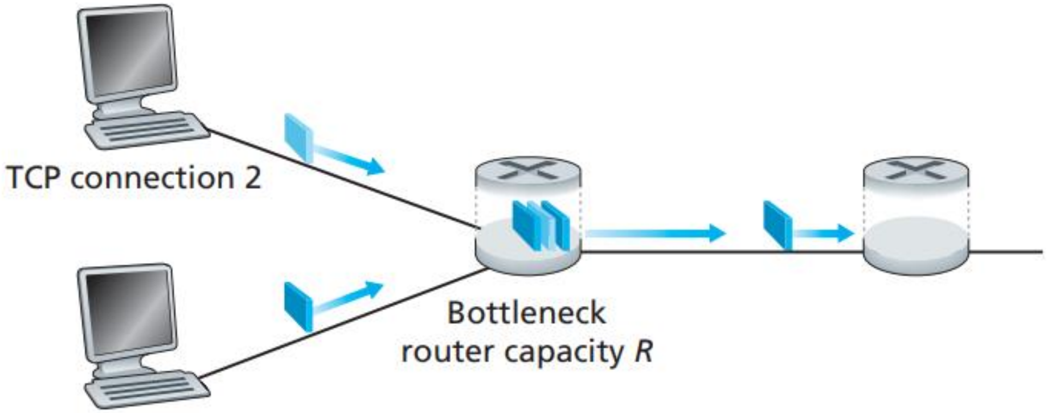


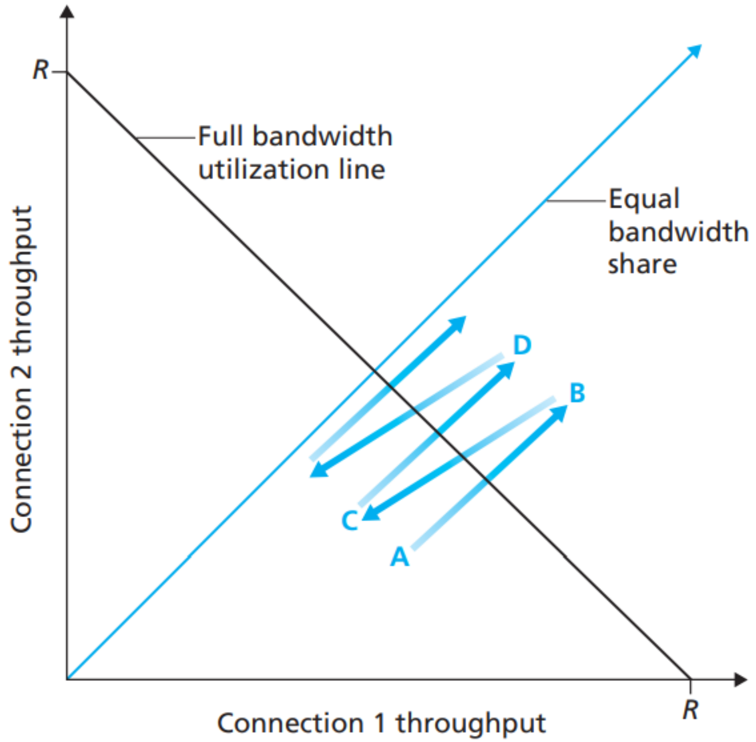
TCP Tahoe vs. TCP Reno

TCP CONGESTION CONTROL: FAIRNESS (1)

- **TCP congestion control** aims to provide **fair utilization** of network.
  - **Fairness goal** – if *K* TCP sessions **share** same link of bandwidth *R*, each should have average **transmission rate** of *R/K*.

- **Example**: Two competing connections.
  - **Additive increase** gives slope of 1 as throughput increases.
  - **Multiplicative decrease** reduces throughput proportionally.





TCP fairness example

TCP CONGESTION CONTROL: FAIRNESS (2)

- **Fairness** and **UDP**:
  - Multimedia apps often do **not** use **TCP**.
    - Do not want sending rate **throttled** by congestion control.
  - Instead they use **UDP**.
    - Send audio/video at **constant** sending rate.
    - Able to **tolerate** packet **loss**.
- **Fairness** and **parallel TCP connections**:
  - Apps can open **multiple parallel TCP connections** between hosts.
    - Web browsers use this to transfer multiple files within web page.
  - **Example:** link of rate *R* with 9 applications, each using one TCP connections:
    - New app requesting **1 TCP connection**, gets rate *R/10*.
    - New app requesting **11 parallel TCP connections**, gets rate *R/2*.

SUMMARY

- Approaches to congestion control.
- TCP congestion control.
- Slow start.
- Congestion avoidance.
- Fast recovery.
- TCP Tahoe vs. TCP Reno.
- Fairness.