Network Transport Layer: TCP Congestion Control

Qiao Xiang, Congming Gao

https://sngroup.org.cn/courses/cnns-xmuf23/index.shtml

11/16/2023

This deck of slides are heavily based on CPSC 433/533 at Yale University, by courtesy of Dr. Y. Richard Yang.

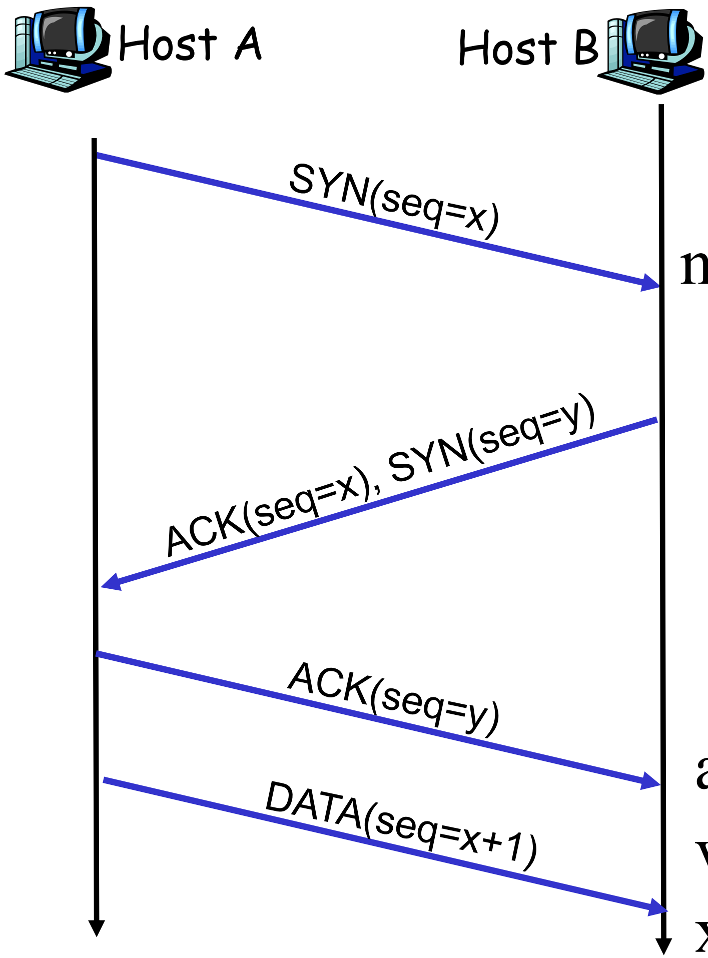
Outline

- Admin and recap
- TCP Congestion Control

Admin

- Lab 3 due on ~~Nov. 19~~ Nov. 22
- Guest lectures (tentative schedule subject to change)
  - 11/21, Yuchao Zhang, BUPT, Traffic Engineering
  - 11/28, Yutong Liu, SJTU, Internet of Things

Recap: Three Way Handshake (TWH) [Tomlinson 1975]



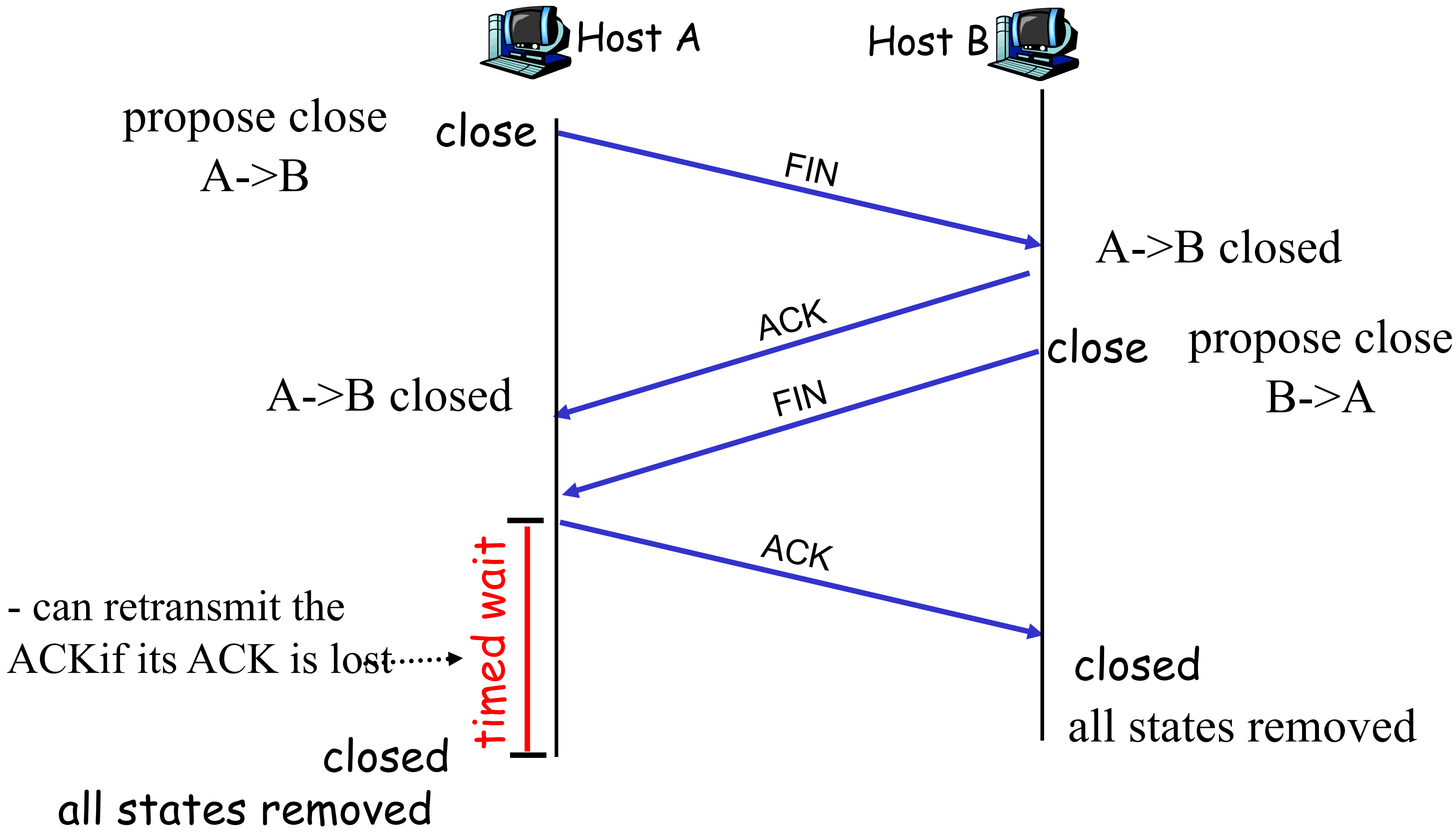
notify initial seq#. Accept?

think of y as a challenge

accept data only after verified y is bounced back x is the init. seq

SYN: indicates connection setup

Recap: TCP Four Way Teardown (For Bi-Directional Transport)

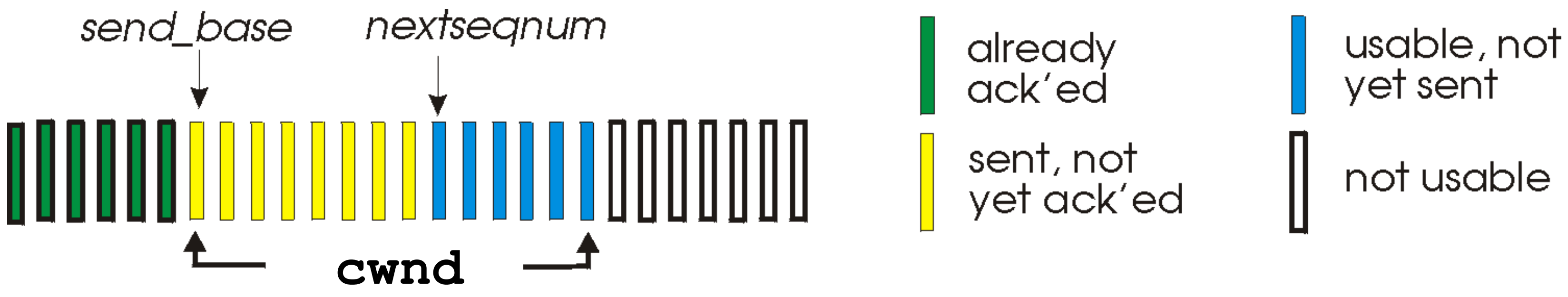


Recap: Transport Design

- Basic structure/reliability: sliding window protocols
- Determine the "right" parameters
  - Timeout
    - mean + variation
  - Sliding window size?

Sliding Window Size Function: Rate Control

- Transmission rate determined by congestion window size, **cwnd**, over segments:



- cwnd segments, each with MSS bytes sent in one RTT:

$$\text{Rate} = \frac{\text{cwnd} * \text{MSS}}{\text{RTT}} \text{ Bytes/sec}$$

Assume W is small enough. Ignore small details. MSS: Minimum Segment Size

Some General Questions

Big picture question:

- How to determine a flow's sending rate?

For better understanding, we need to look at a few basic questions:

- What is congestion (cost of congestion)?
- Why are desired properties of congestion control?

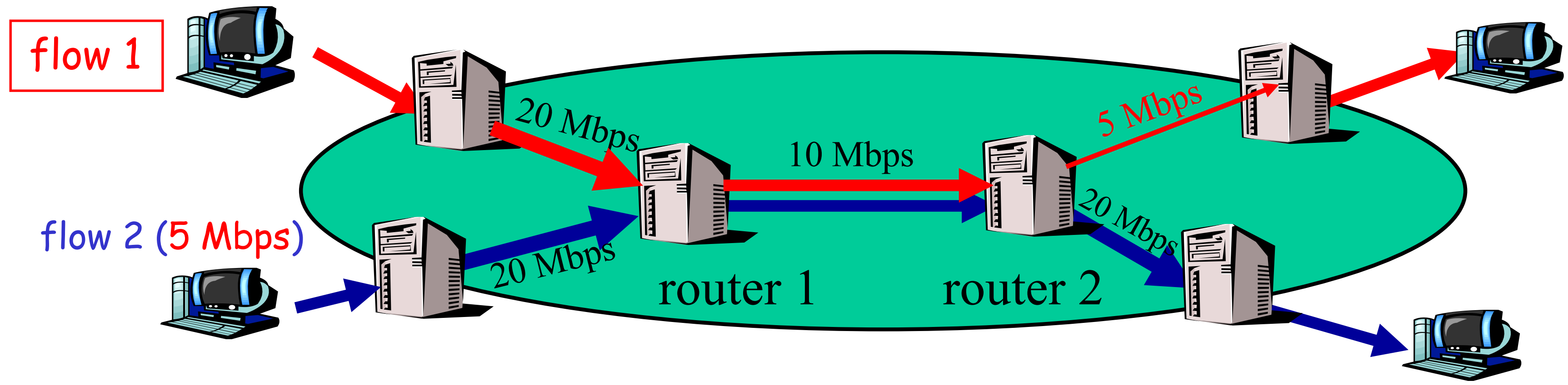
Roadmap

- What is congestion
- The basic CC alg
- TCP/reno CC
- TCP/Vegas
- A unifying view of TCP/Reno and TCP/Vegas
- Network wide resource allocation
  - Framework
  - Axiom derivation of network-wide objective function
  - Derive distributed algorithm

Outline

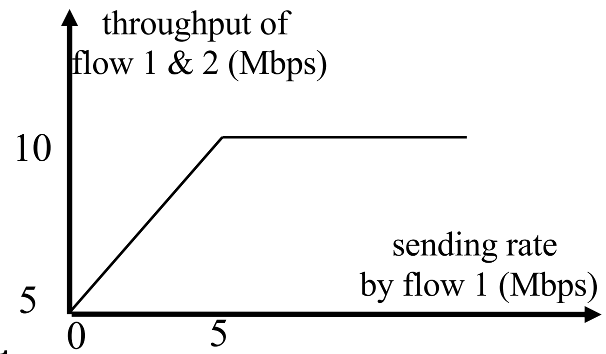
- Admin and recap
- TCP Reliability
- Transport congestion control
  - *what is congestion (cost of congestion)*

Cause/Cost of Congestion: Single Bottleneck

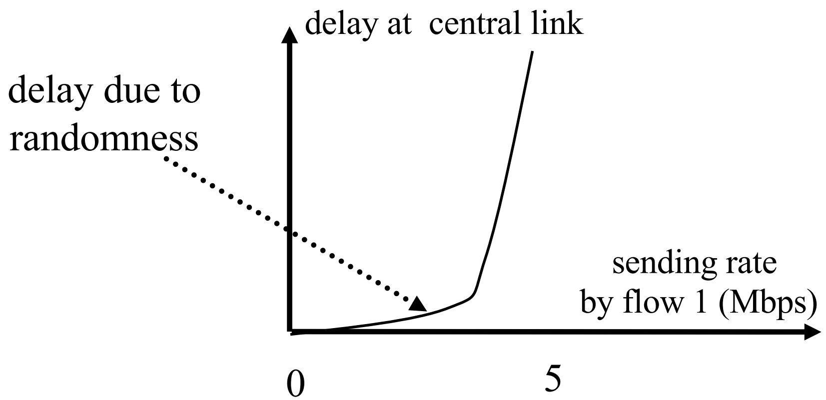


- Flow 2 has a fixed sending rate of 5 Mbps
- We vary the sending rate of flow 1 from 0 to 20 Mbps
- Assume
  - no retransmission; link from router 1 to router 2 has infinite buffer

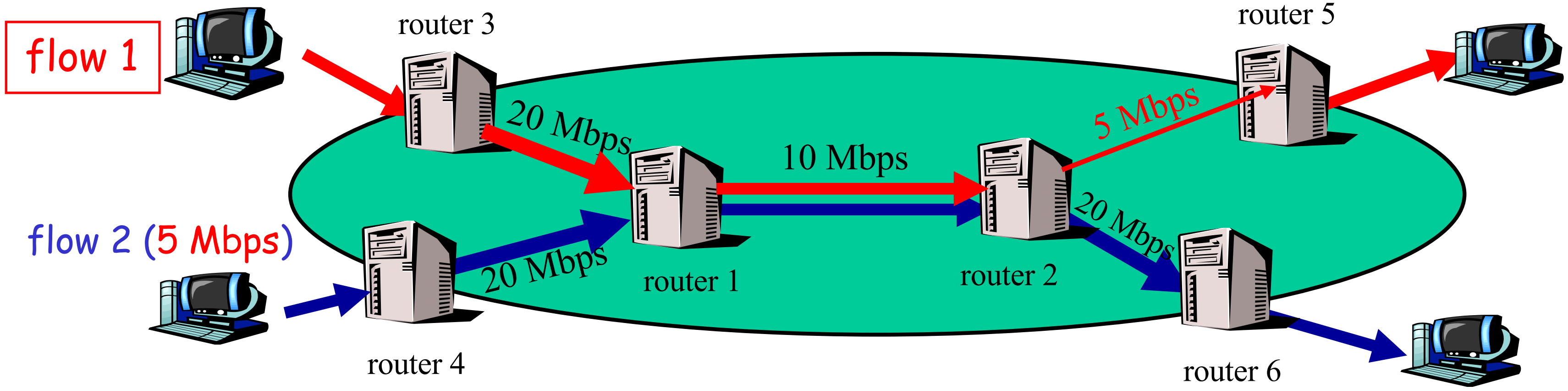
throughput: e2e packets delivered in unit time



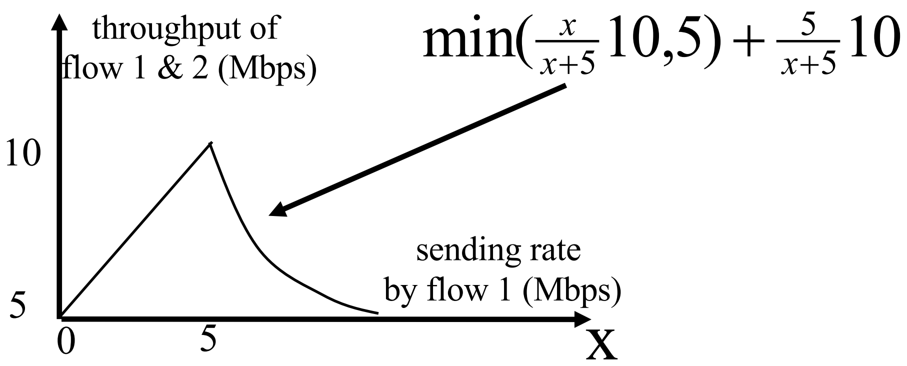
Delay?



Cause/Cost of Congestion: Single Bottleneck



- Assume
  - no retransmission
  - the link from router 1 to router 2 has finite buffer
  - throughput: e2e packets delivered in unit time



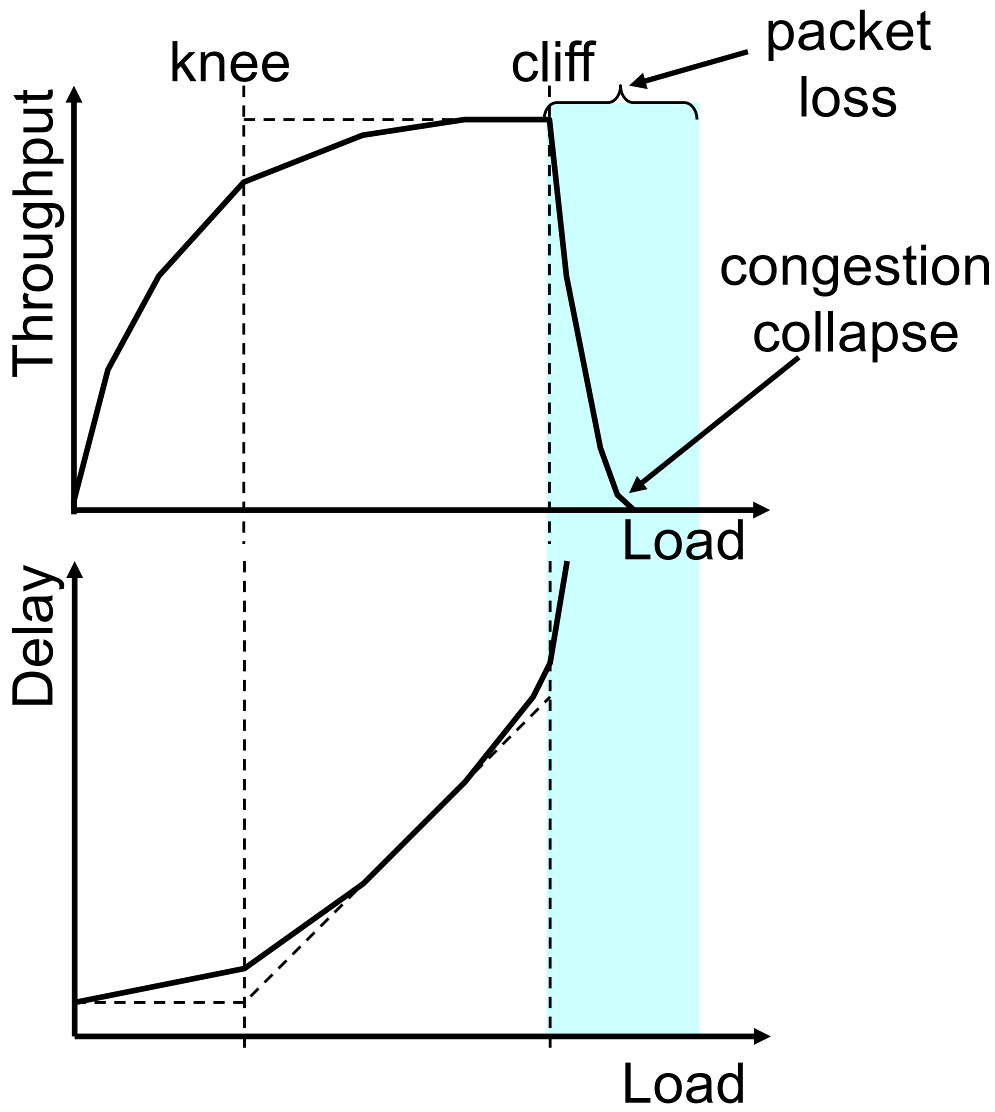
- Zombie packet: a packet dropped at the link from router 2 to router 5; the upstream transmission from router 1 to router 2 used for that packet was wasted!

Summary: The Cost of Congestion

When sources sending rate too high for the *network* to handle":

- Packet loss =>
  - wasted upstream bandwidth when a pkt is discarded at downstream
  - wasted bandwidth due to retransmission (a pkt goes through a link multiple times)

- High delay



Outline

- Admin and recap
- TCP Reliability
- Transport congestion control
  - what is congestion (cost of congestion)
  - *basic congestion control alg.*

Rate-based vs. Window-based

Rate-based:

- Congestion control by explicitly controlling the sending rate of a flow, e.g., set sending rate to 128Kbps
- Example: ATM

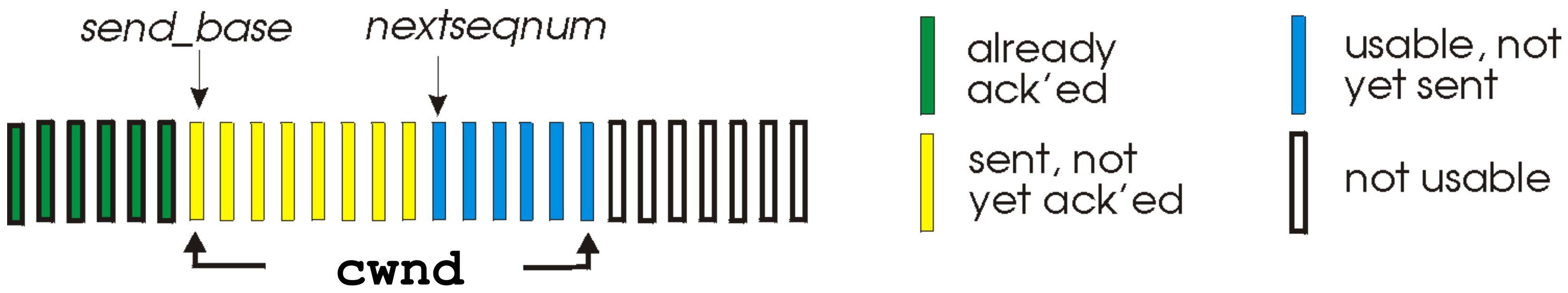
Window-based:

- Congestion control by controlling the window size of a sliding window, e.g., set window size to 64KBytes
- Example: TCP

Discussion: rate-based vs. window-based

Sliding Window Size Function: Rate Control

- Transmission rate determined by congestion window size, **cwnd**, over segments:

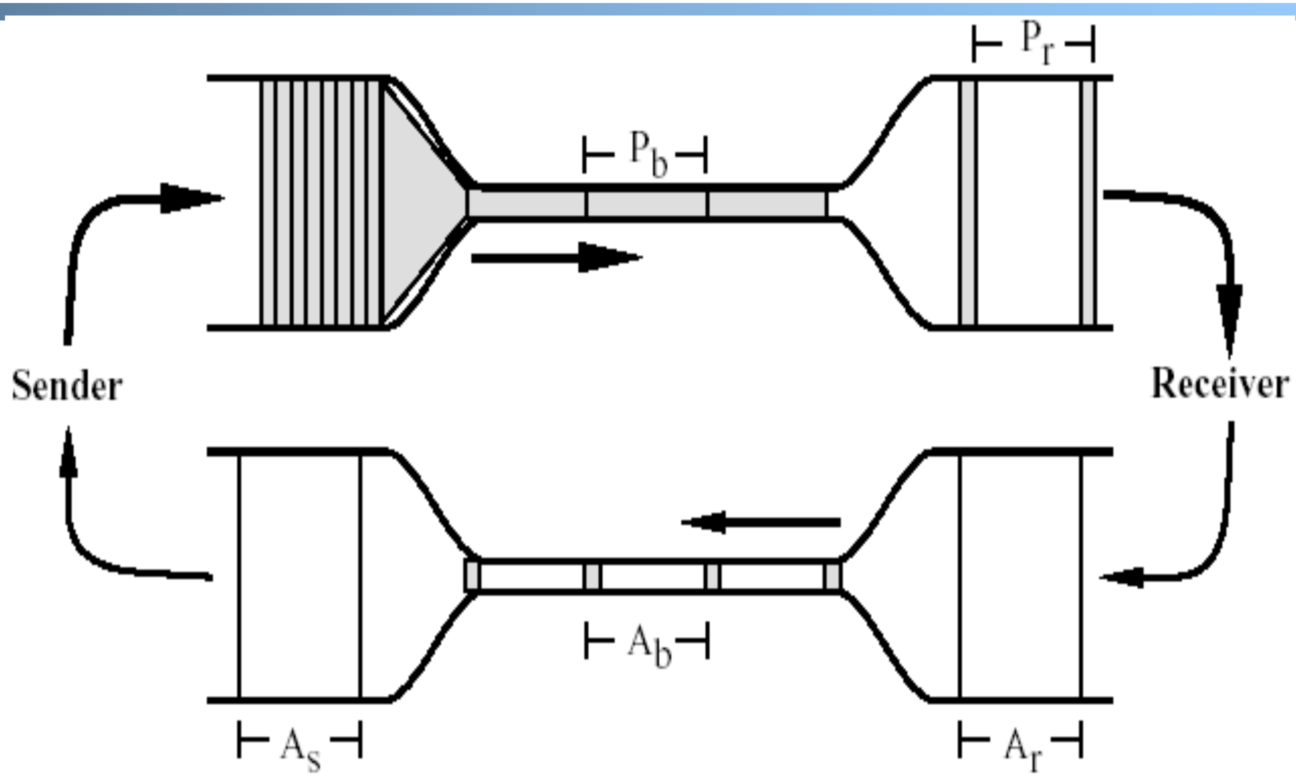


- cwnd segments, each with MSS bytes sent in one RTT:

$$\text{Rate} = \frac{\text{cwnd} * \text{MSS}}{\text{RTT}} \text{ Bytes/sec}$$

Assume W is small enough. Ignore small details. MSS: Maximum Segment Size

Window-based Congestion Control



- Window-based congestion control is self-clocking: considers flow conservation, and adjusts to RTT variation automatically.
- Hence, for better safety, more designs use window-based design.

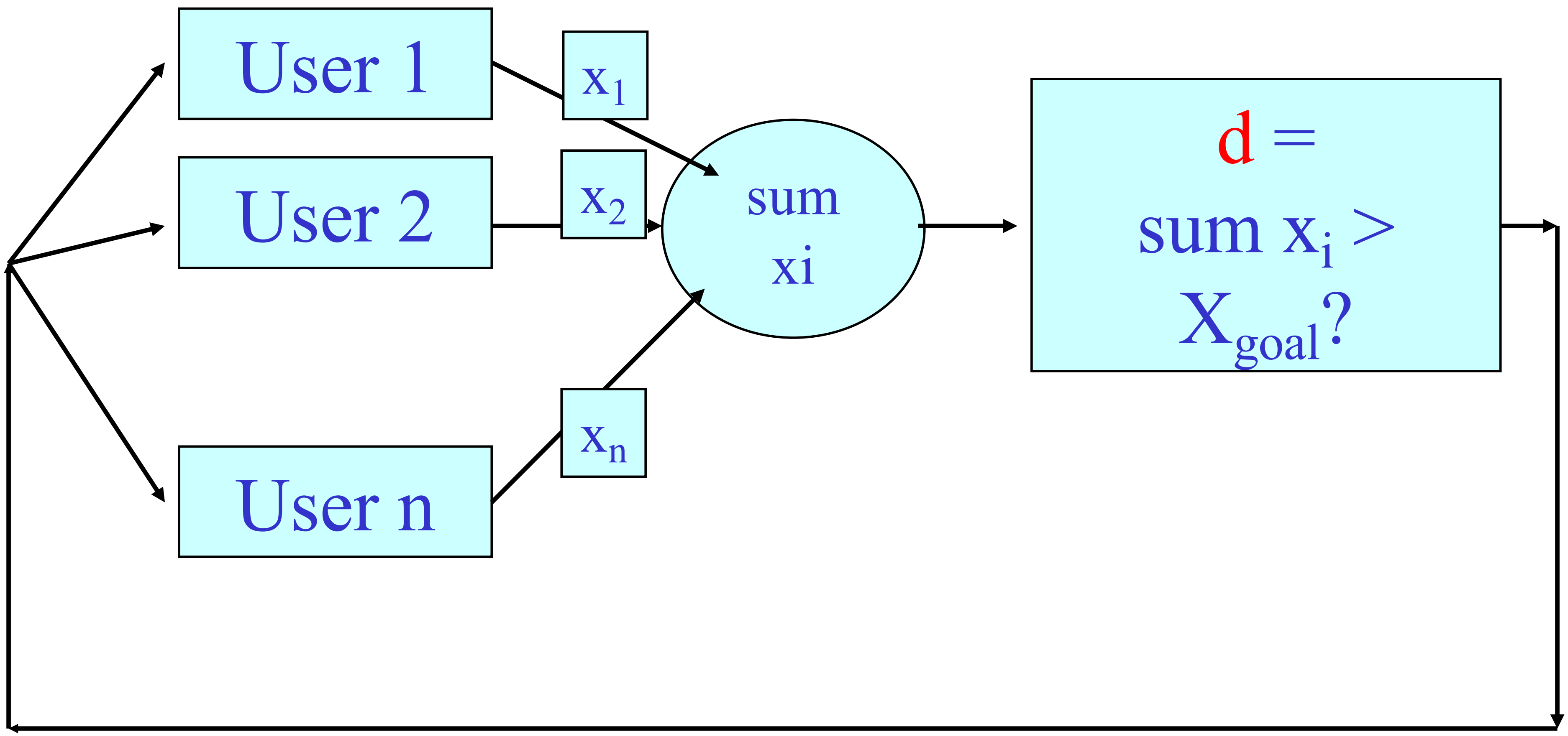
The Desired Properties of a Congestion Control Scheme

- Efficiency: close to full utilization but low delay
  - fast convergence after disturbance

- Fairness (resource sharing)

- Distributedness (no central knowledge for scalability)

Derive CC: A Simple Model



Flows observe congestion signal d, and locally take actions to adjust rates.

Linear Control

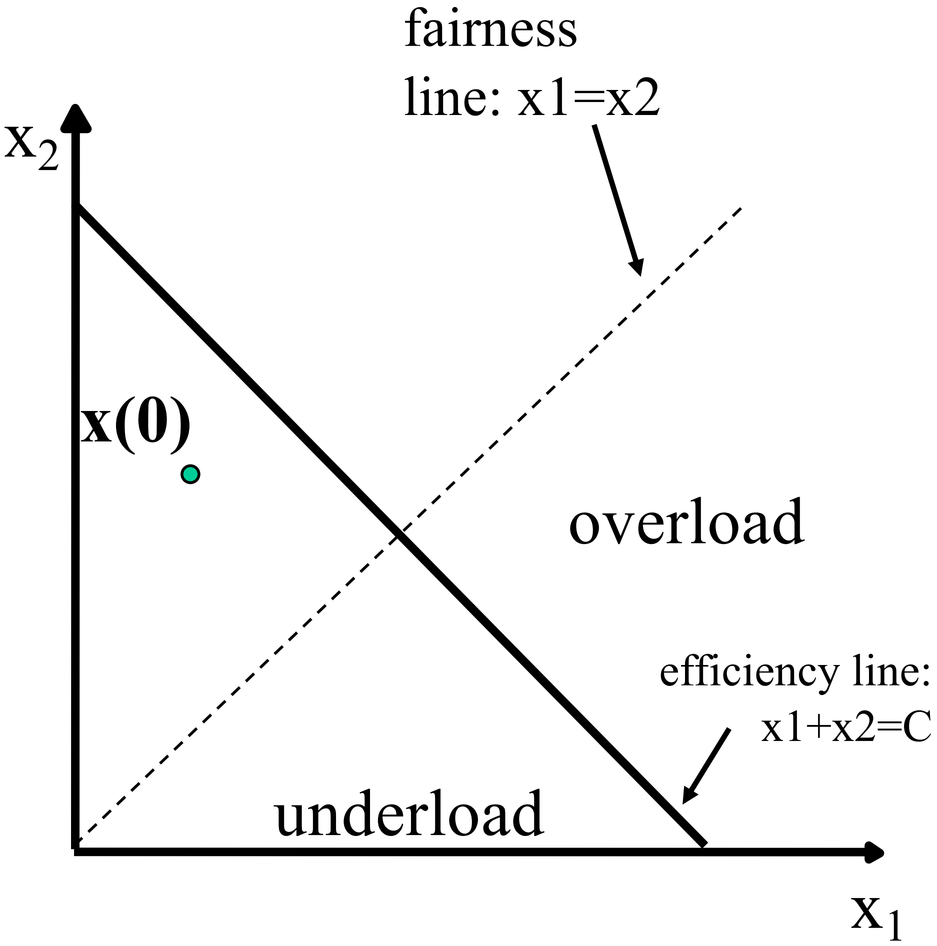
- Proposed by Chiu and Jain (1988)
- The simplest control strategy

$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{if } \mathrm{d(t) = no\ cong.} \\ a_D + b_D x_i(t) & \text{if } \mathrm{d(t) = cong.} \end{cases}$$

Discussion: values of the parameters?

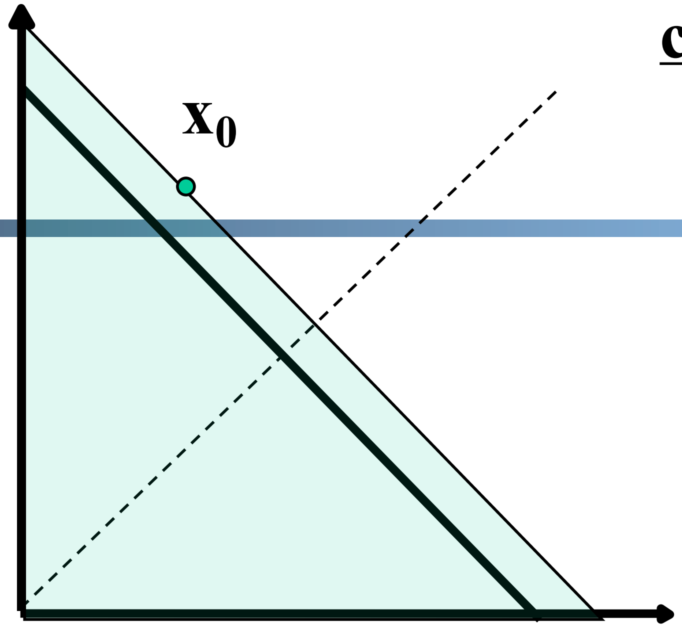
State Space of Two Flows

$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{if } \mathrm{d(t) = no\ cong.} \\ a_D + b_D x_i(t) & \text{if } \mathrm{d(t) = cong.} \end{cases}$$

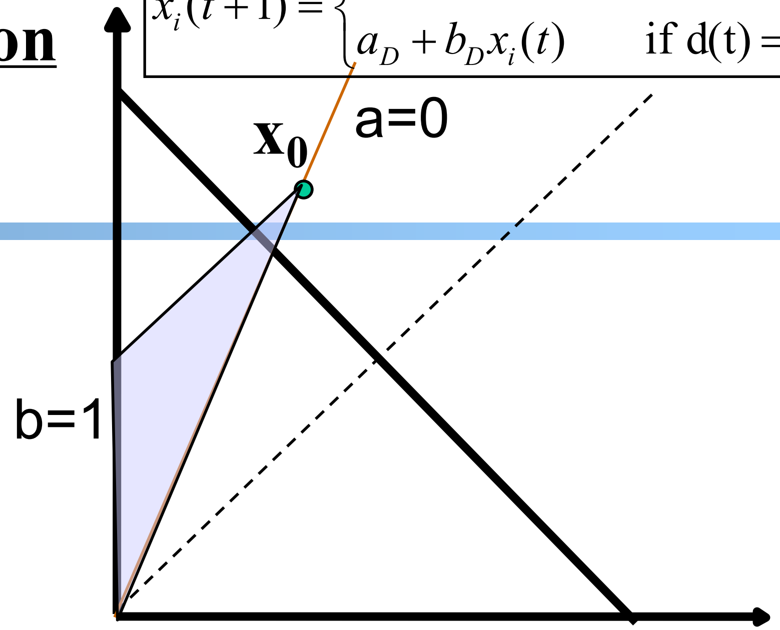


congestion

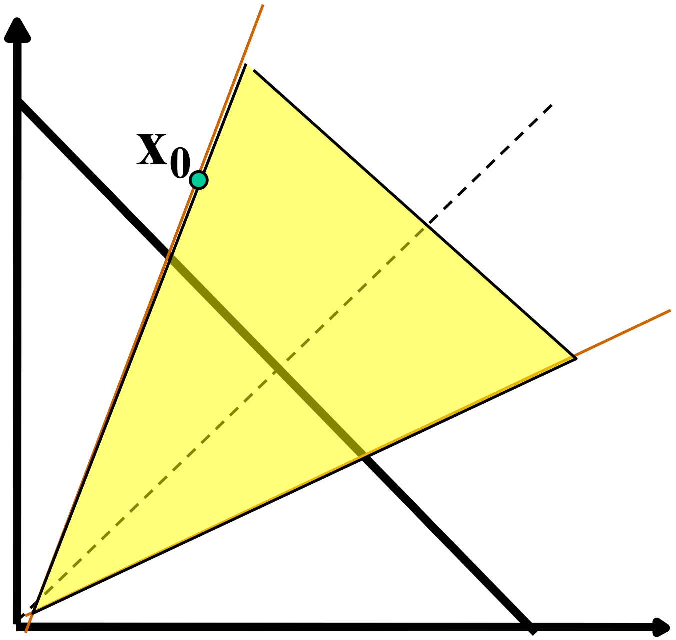
$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{if } d(t) = \text{no cong.} \\ a_D + b_D x_i(t) & \text{if } d(t) = \text{cong.} \end{cases}$$



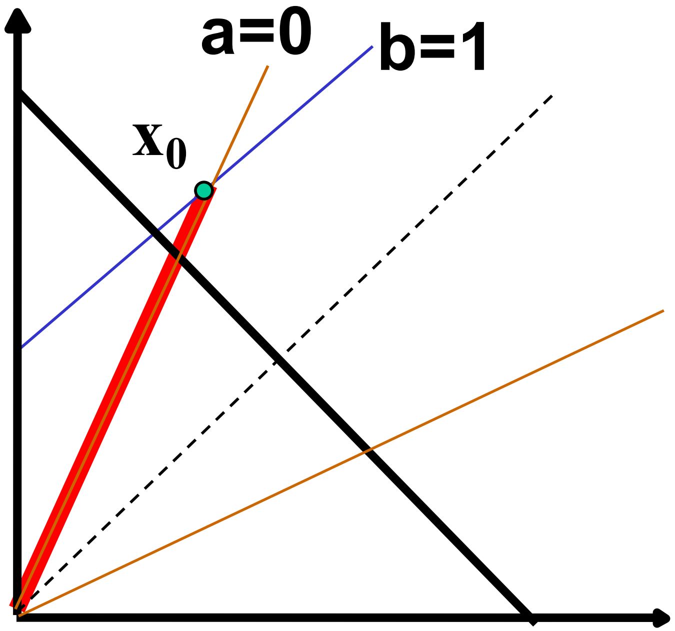
efficiency



efficiency: distributed linear rule



fairness



intersection

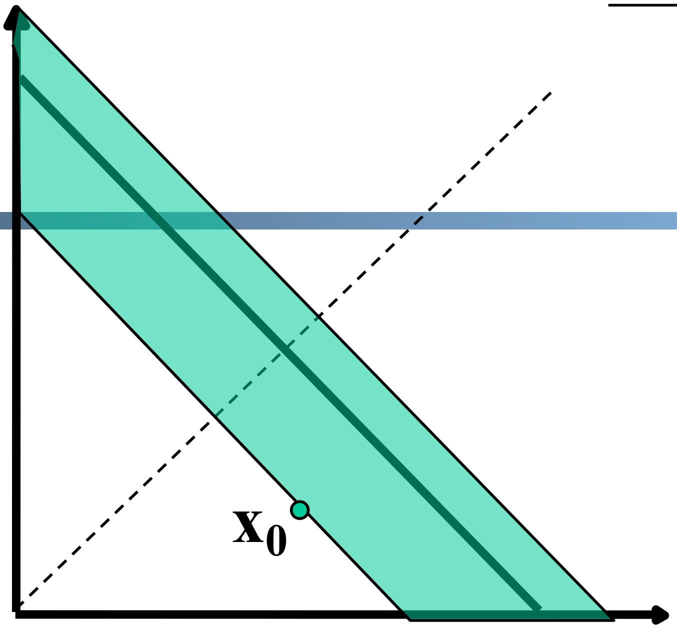
Implication: Congestion (overload) Case

- In order to get closer to efficiency and fairness after each update, decreasing of rate must be multiplicative decrease (MD)
  - $a_D = 0$
  - $b_D < 1$

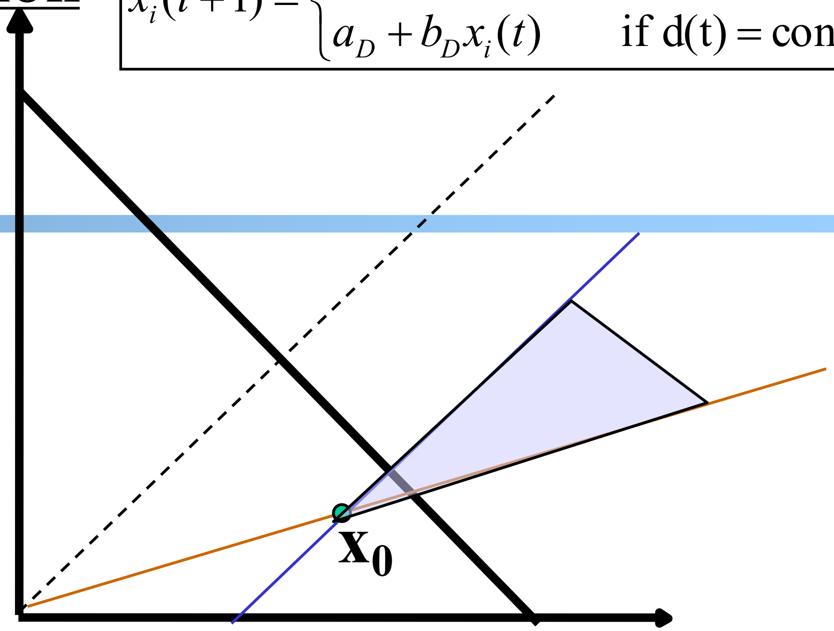
$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{if } \mathrm{d(t) = no\ cong.} \\ b_D x_i(t) & \text{if } \mathrm{d(t) = cong.} \end{cases}$$

no-congestion

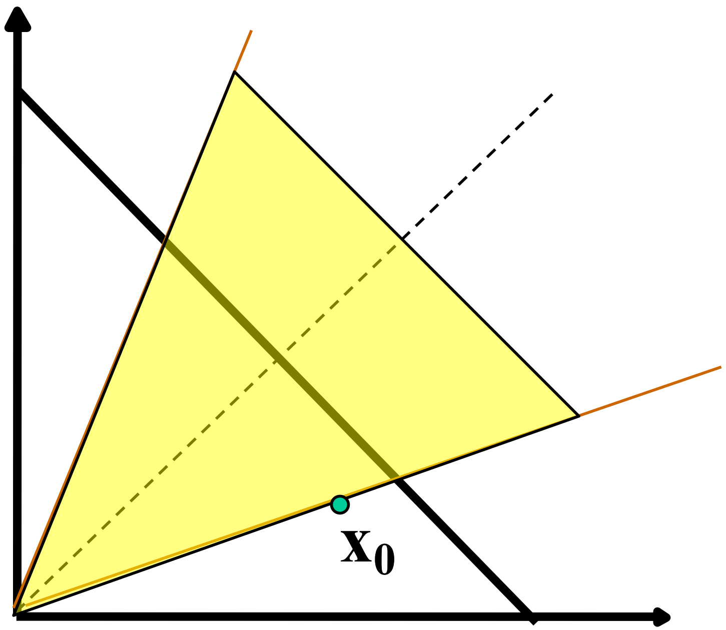
$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{if } d(t) = \text{no cong.} \\ a_D + b_D x_i(t) & \text{if } d(t) = \text{cong.} \end{cases}$$



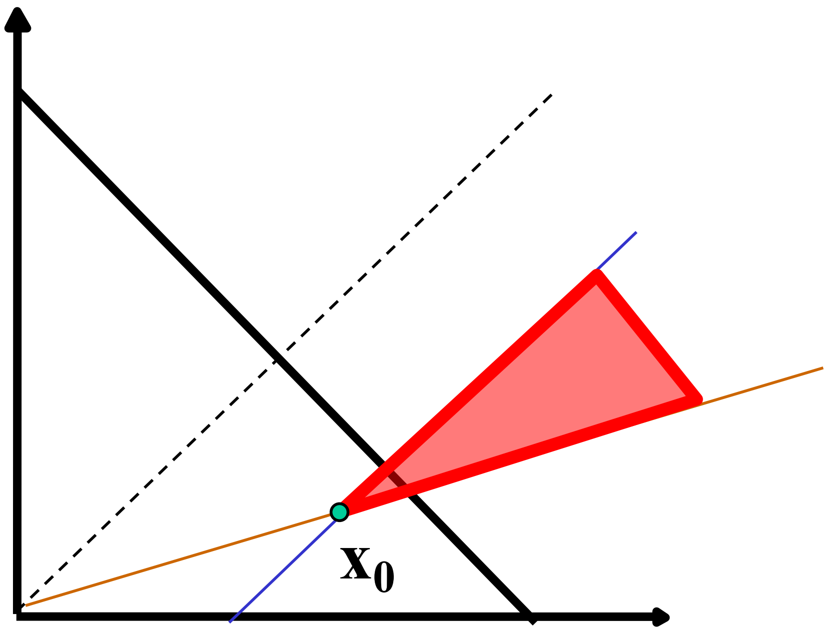
efficiency



efficiency: distributed linear rule



fairness



convergence

Implication: No Congestion Case

- In order to get closer to efficiency and fairness after each update, additive and multiplicative increasing (AMI), i.e.,
  - $a_I > 0$, $b_I > 1$

$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{if } \mathrm{d(t) = no\ cong.} \\ b_D x_i(t) & \text{if } \mathrm{d(t) = cong.} \end{cases}$$

- Simply additive increase gives better improvement in fairness (i.e., getting closer to the fairness line)
- Multiplicative increase may grow faster

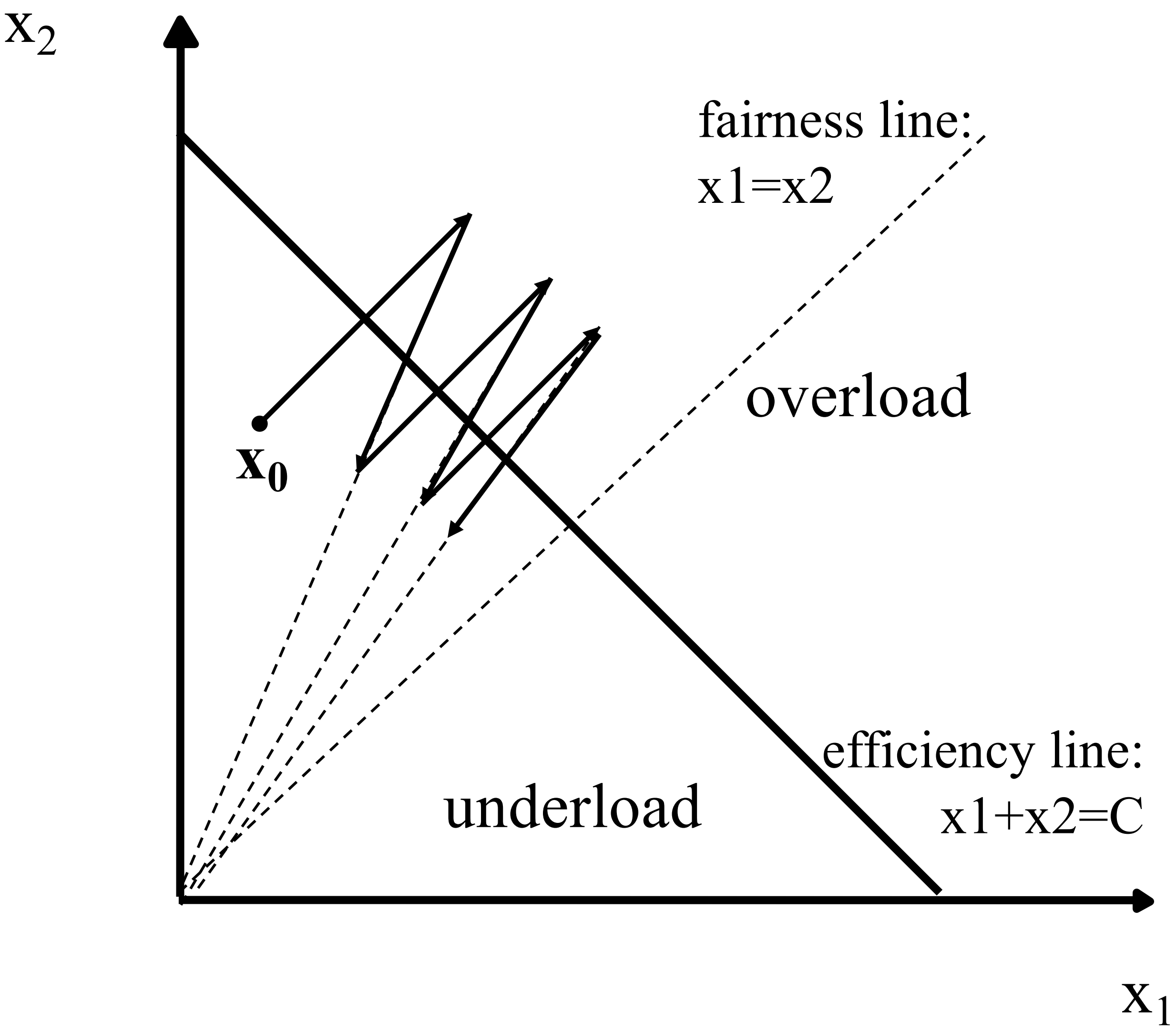
Intuition: State Trace Analysis of Four Special Cases

	Additive Decrease	Multiplicative Decrease
Additive Increase	AIAD ($b_I=b_D=1$)	AIMD ($b_I=1$, $a_D=0$)
Multiplicative Increase	MIAD ($a_I=0$, $b_I>1$, $b_D=1$)	MIMD ($a_I=a_D=0$)

$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{if } d(t) = \text{no cong.} \\ a_D + b_D x_i(t) & \text{if } d(t) = \text{cong.} \end{cases}$$

Discussion: state transition trace.

AIMD: State Transition Trace



Intuition: Another Look

- Consider the difference or ratio of the rates of two flows
  - AIAD
    - difference does not change
  - MIMD
    - ratio does not change
  - MIAD
    - difference becomes bigger
  - AIMD
    - difference does not change

Outline

- Admin and recap
- TCP Reliability
- Transport congestion control
  - what is congestion (cost of congestion)
  - basic congestion control alg.
  - *TCP/reno congestion control*

TCP Congestion Control

- Closed-loop, end-to-end, window-based congestion control
- Designed by Van Jacobson in late 1980s, based on the AIMD alg. of Dah-Ming Chu and Raj Jain
- Worked in a large range of bandwidth values: the bandwidth of the Internet has increased by more than 200,000 times
- Many versions
  - TCP/Tahoe: this is a less optimized version
  - TCP/Reno: many OSs today implement Reno type congestion control
  - TCP/Vegas: not currently used

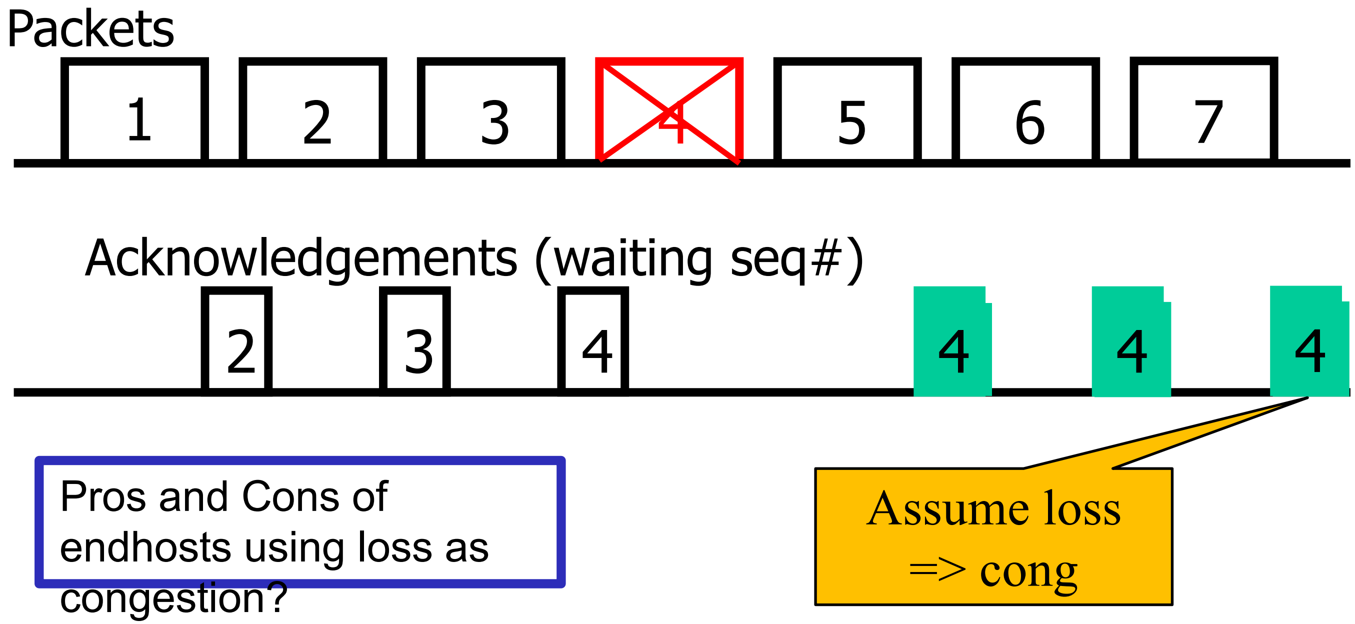
For more details: see TCP/IP illustrated; or read http://lxr.linux.no/source/net/ipv4/tcp_input.c for linux implementation

Mapping A(M)I-MD to Protocol

- Basic questions to look at:
  - How to obtain d(t)--the congestion signal?
  - What values do we choose for the formula?
  - How to map formula to code?

$$x_i(t+1) = \begin{cases} a_I + x_i(t) & \text{if } \mathrm{d(t) = no\ cong.} \\ b_D x_i(t) & \text{if } \mathrm{d(t) = cong.} \end{cases}$$

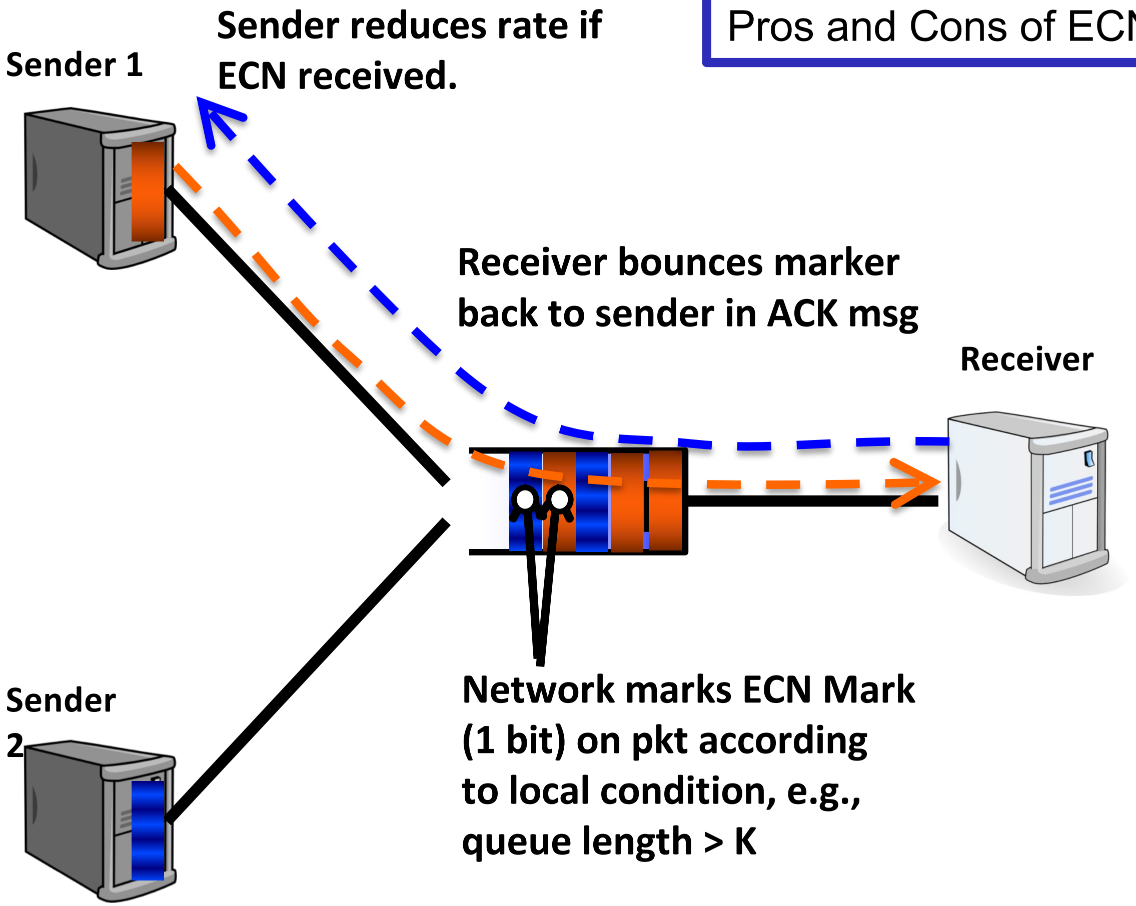
Obtain d(t) Approach 1: End Hosts Consider Loss as Congestion



Pros and Cons of endhosts using loss as congestion?

Obtain d(t) Approach 2: Network Feedback (ECN: Explicit Congestion Notification)

Pros and Cons of ECN?



Mapping A(M)I-MD to Protocol

- Basic questions to look at:
  - How to obtain d(t)--the congestion signal?
  - What values do we choose for the formula?
  - How to map formula to code?

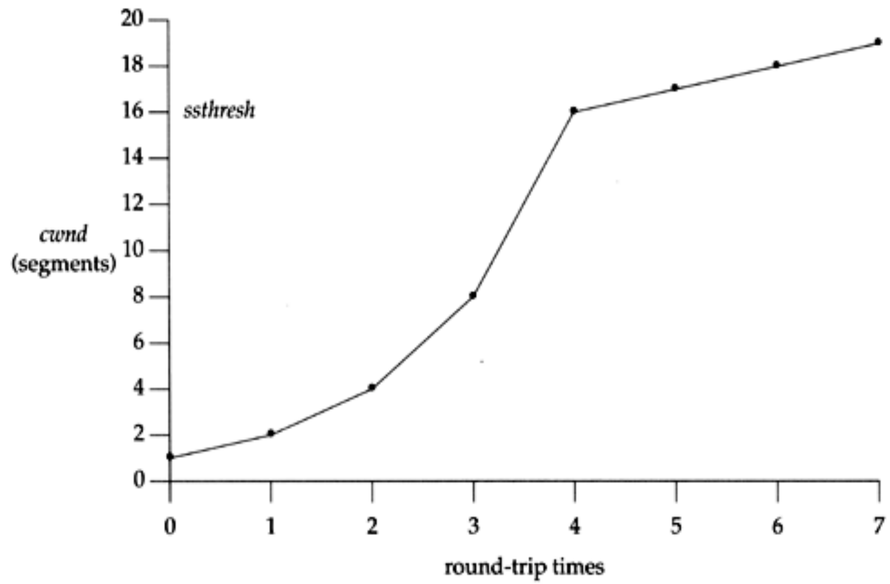
$$x_i(t+1) = \begin{cases} a_I + x_i(t) & \text{if } \mathrm{d(t)} = \text{no cong.} \\ b_D x_i(t) & \text{if } \mathrm{d(t)} = \text{cong.} \end{cases}$$

TCP/Reno Formulas

- Multiplicative Increase (MI)
  - double *the rate:* $x(t+1) = 2\, x(t)$
- Additive Increase (AI)
  - Linear increase *the rate:* $x(t+1) = x(t) + 1$
- Multiplicative decrease (MD)
  - half *the rate:* $x(t+1) = 1/2\, x(t)$

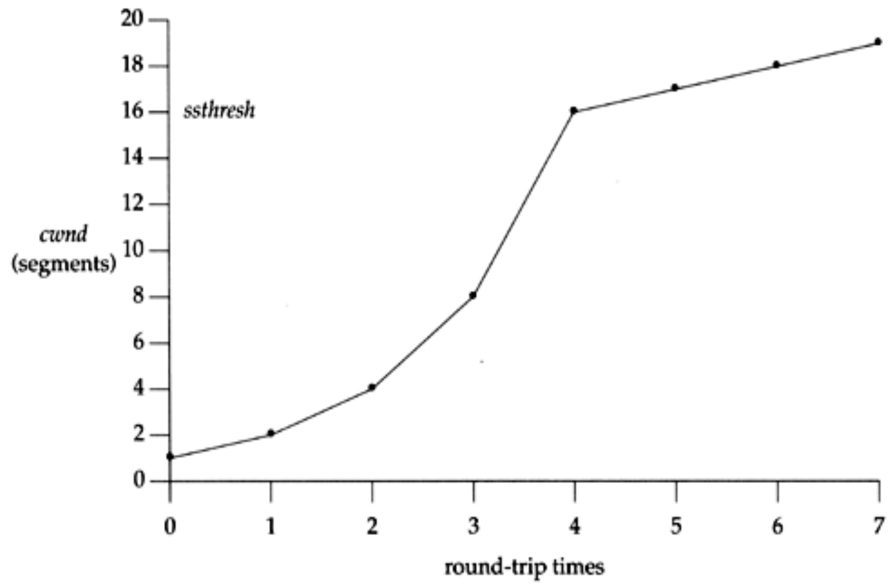
TCP/Reno Formula Switching (Control Structure)

- Two "phases"
  - slow-start
    - Goal: getting to equilibrium gradually but quickly, to get a rough estimate of the optimal of *cwnd*
    - *Formula: MI*
  - congestion avoidance
    - Goal: Maintains equilibrium and reacts around equilibrium
    - Formula: AI MD



TCP/Reno Formula Switching (Control Structure)

- Important variables:
  - **cwnd**: congestion window size
  - **ssthresh**: threshold between the slow-start phase and the congestion avoidance phase
- If cwnd < ssthresh
  - MI
- Else
  - AIMD



MI: Slow Start

- Algorithm: MI
  - double *cwnd* every RTT until network congested

- Goal: getting to equilibrium gradually but quickly, to get a rough estimate of the optimal of *cwnd*

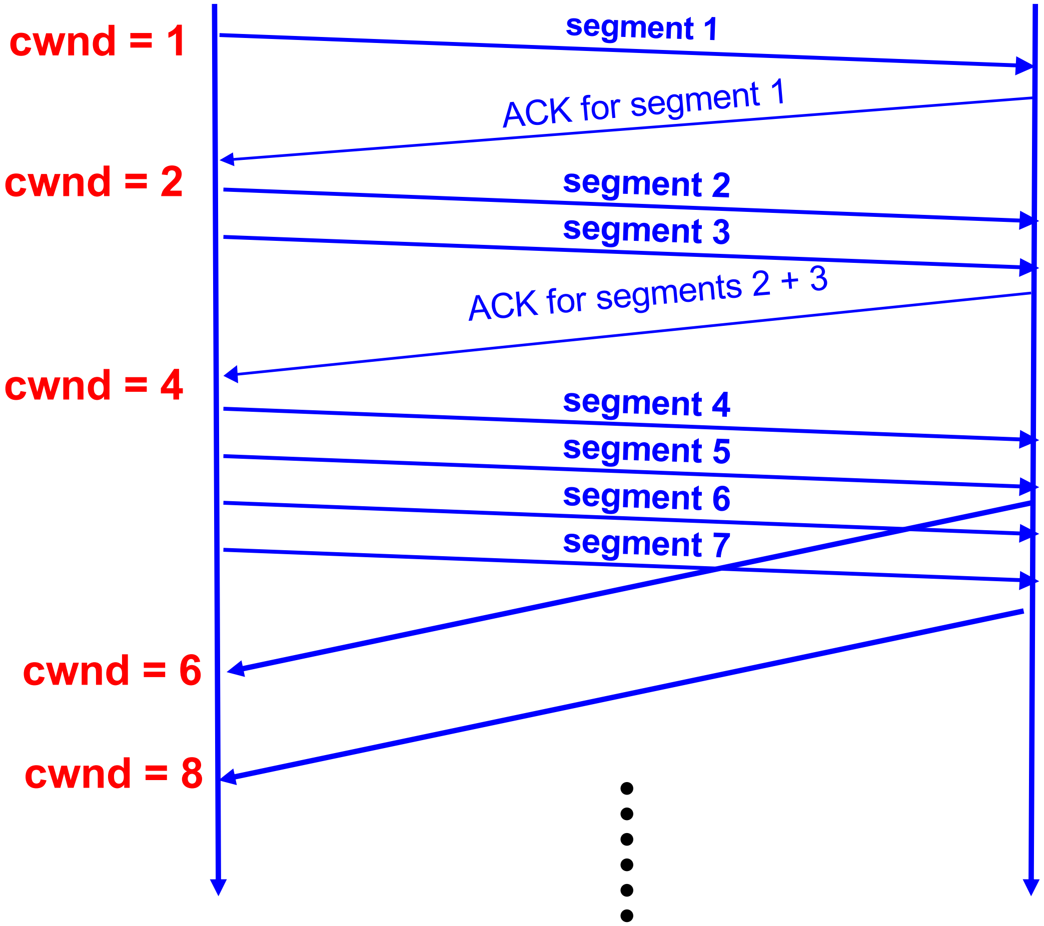
MI: Slow-start

Initially:

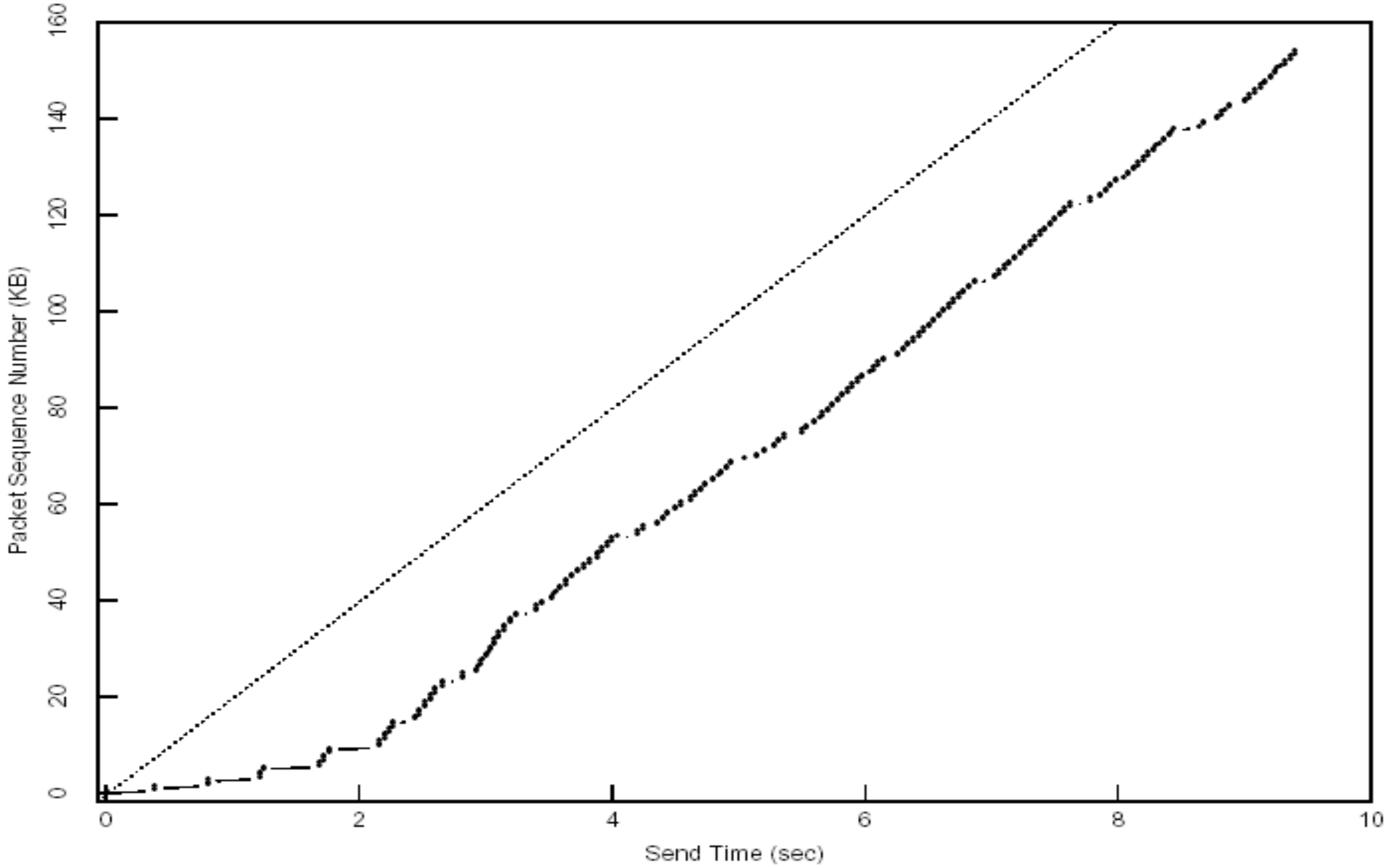
cwnd = 1;
ssthresh = infinite (e.g., 64K);

For each newly ACKed segment:

if (cwnd < ssthresh)
/* MI: slow start*/
cwnd = cwnd + 1;



Startup Behavior with Slow-start



See [Jac89]

AIMD: Congestion Avoidance

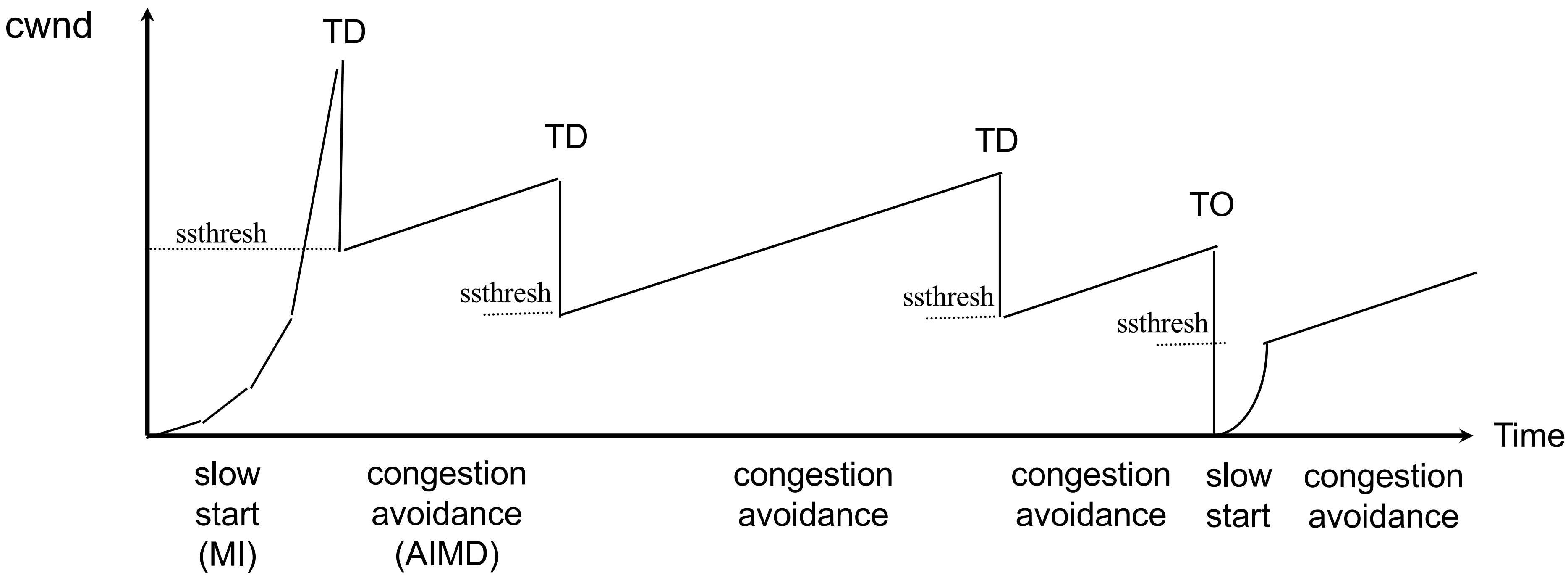
- Algorithm: AIMD
  - increases window by 1 per round-trip time (how?)
  - cuts window size
    - to half when detecting congestion by 3DUP
    - to 1 if timeout
    - if already timeout, doubles timeout

- Goal: Maintains equilibrium and reacts around equilibrium

TCP/Reno Full Alg

```
Initially:
    cwnd = 1;
    ssthresh = infinite (e.g., 64K);
For each newly ACKed segment:
    if (cwnd < ssthresh)          // slow start: MI
        cwnd = cwnd + 1;
    else
                                  // congestion avoidance; AI
        cwnd += 1/cwnd;
Triple-duplicate ACKs:
                                  // MD
    cwnd = ssthresh = cwnd/2;
Timeout:
    ssthresh = cwnd/2;           // reset
    cwnd = 1;
(if already timed out, double timeout value; this is called exponential backoff)
```

TCP/Reno: Big Picture



TD: Triple duplicate acknowledgements
TO: Timeout