Network Transport Layer: TCP Connection Management, Congestion Control

Qiao Xiang

https://qiaoxiang.me/courses/cnns-xmuf21/index.shtml

11/18/2021

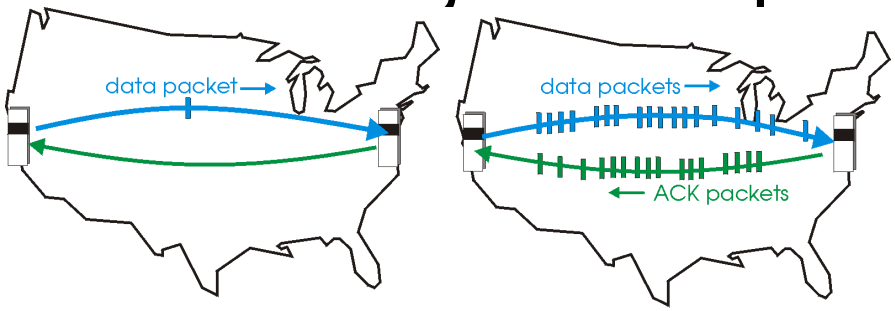
This deck of slides are heavily based on CPSC 433/533 at Yale University, by courtesy of Dr. Y. Richard Yang.

Admin

- Lab assignment 5 to be posted tomorrow
- Class Project
  - 15% of your final score
  - Please start ASAP
  - Talk to the instructor or TA to get feedback
  - Send your topic and team member list to TA

Recap: Reliable Transport

- Basic structure: sliding window protocols



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

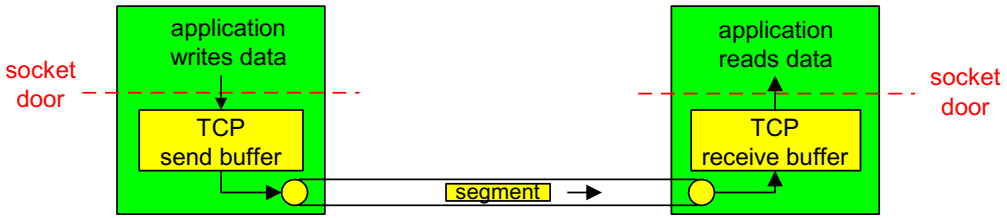
General technique: pipelining.

- Realization: GBN or SR

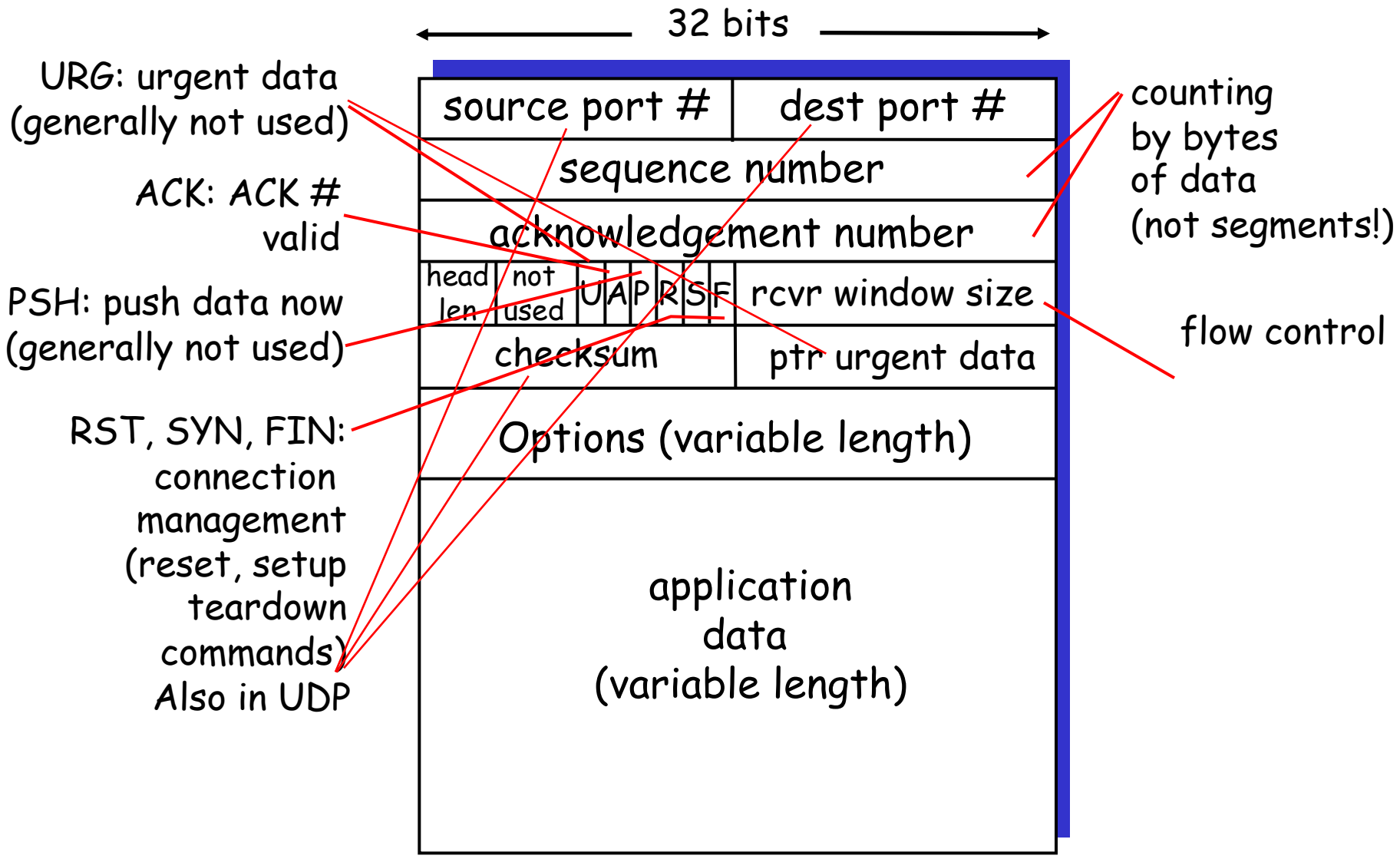
	Go-back-n	Selective Repeat
data bandwidth: sender to receiver (avg. number of times a pkt is transmitted)	Less efficient $\frac{1-p+pw}{1-p}$	More efficient $\frac{1}{1-p}$
ACK bandwidth (receiver to sender)	More efficient	Less efficient
Relationship between M (the number of seq#) and W (window size)	$M > W$	$M \geq 2W$
Buffer size at receiver	1	W
Complexity	Simpler	More complex

Recap: TCP Reliable Data Transfer

- Connection-oriented:
  - connection management
    - setup (exchange of control msgs) init's sender, receiver state before data exchange
    - close
- Full duplex data:
  - bi-directional data flow in same connection
- A sliding window protocol
  - a combination of go-back-n and selective repeat:
    - send & receive buffers
    - cumulative acks
    - TCP uses a single retransmission timer
    - do not retransmit all packets upon timeout



Recap: TCP Segment Structure



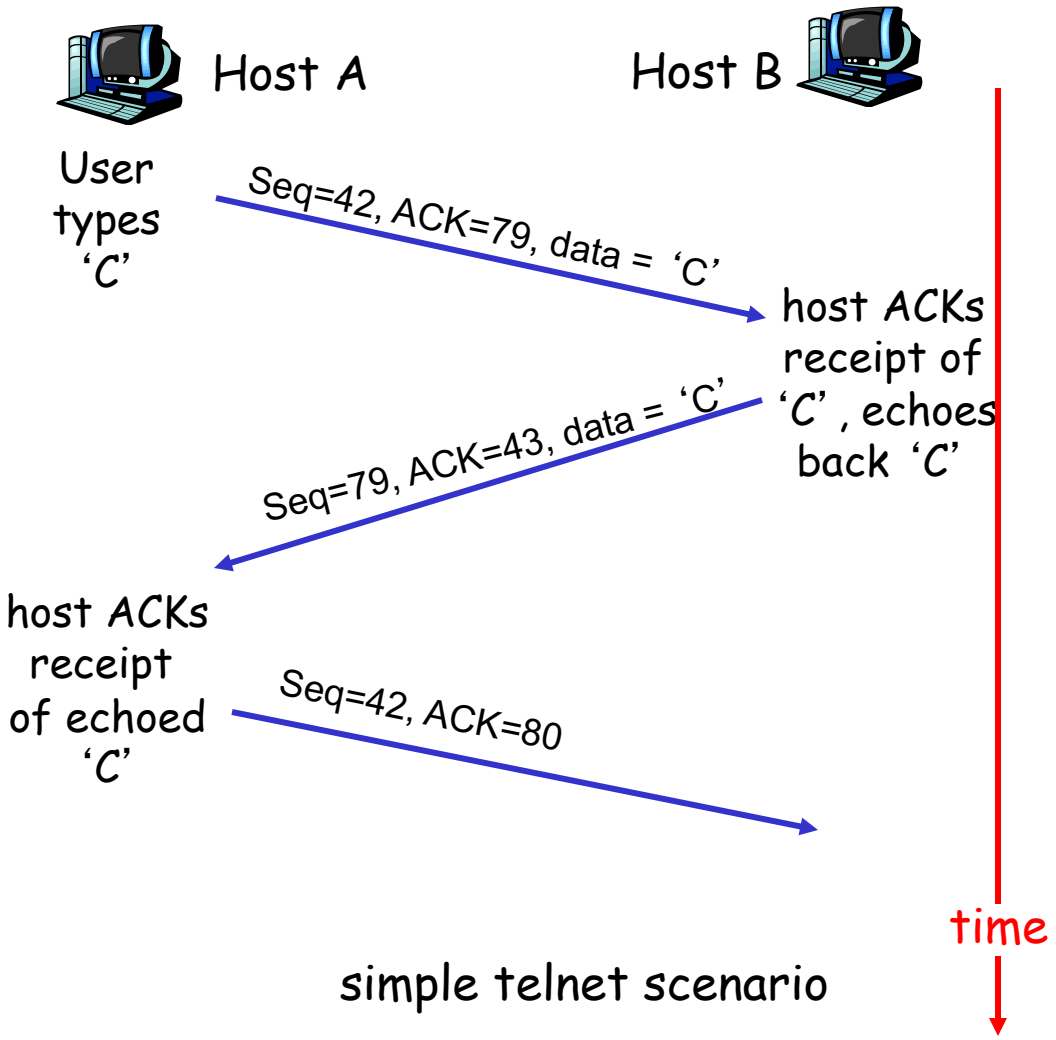
Recap: TCP Seq. #'s and ACKs

Seq. #'s:

- byte stream "number" of first byte in segment's data

ACKs:

- seq # of next byte expected from other side
- cumulative ACK in standard header
- selective ACK in options



simple telnet scenario

Recap: TCP Reliable Data Transfer

- Basic structure: sliding window protocol
- Remaining issue: How to determine the "right" parameters?
  - ✓ timeout value

```
Timeout = EstRTT + 4 * DevRTT

EstRTT = (1-alpha)*EstRTT + alpha*SampleRTT
```

  - sliding window size?

Recap: Fast Retransmit

- Issue: Timeout period often relatively long:
  - long delay before resending lost packet
- Question: Can we detect loss faster than RTT?

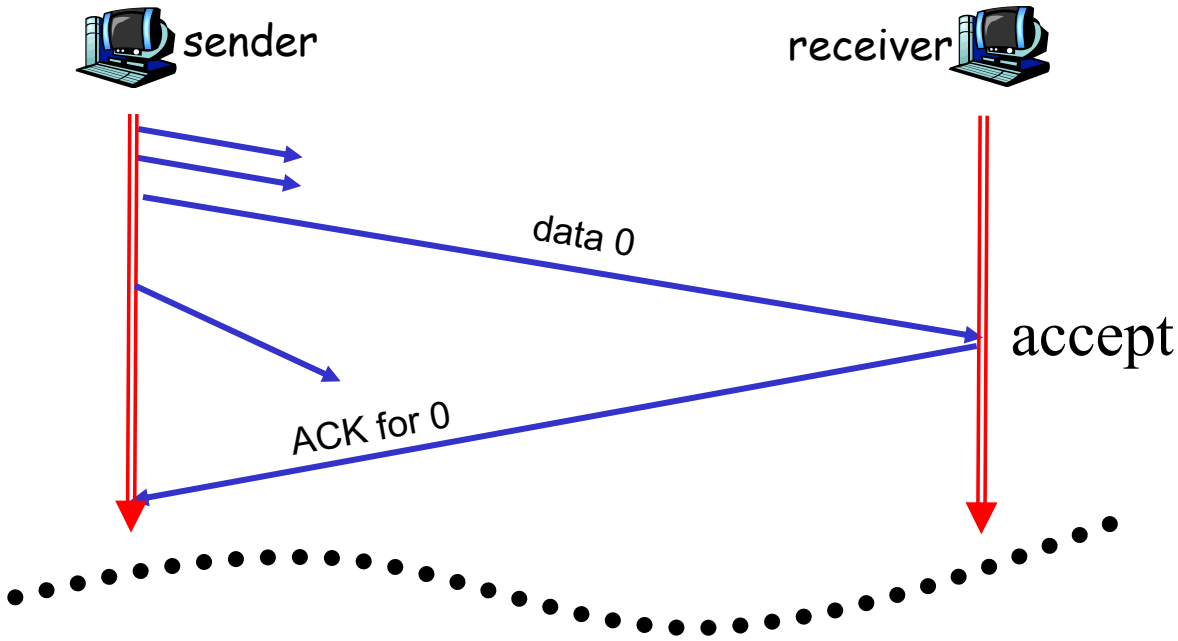
- Detect lost segments via duplicate ACKs
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs

- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - resend segment before timer expires

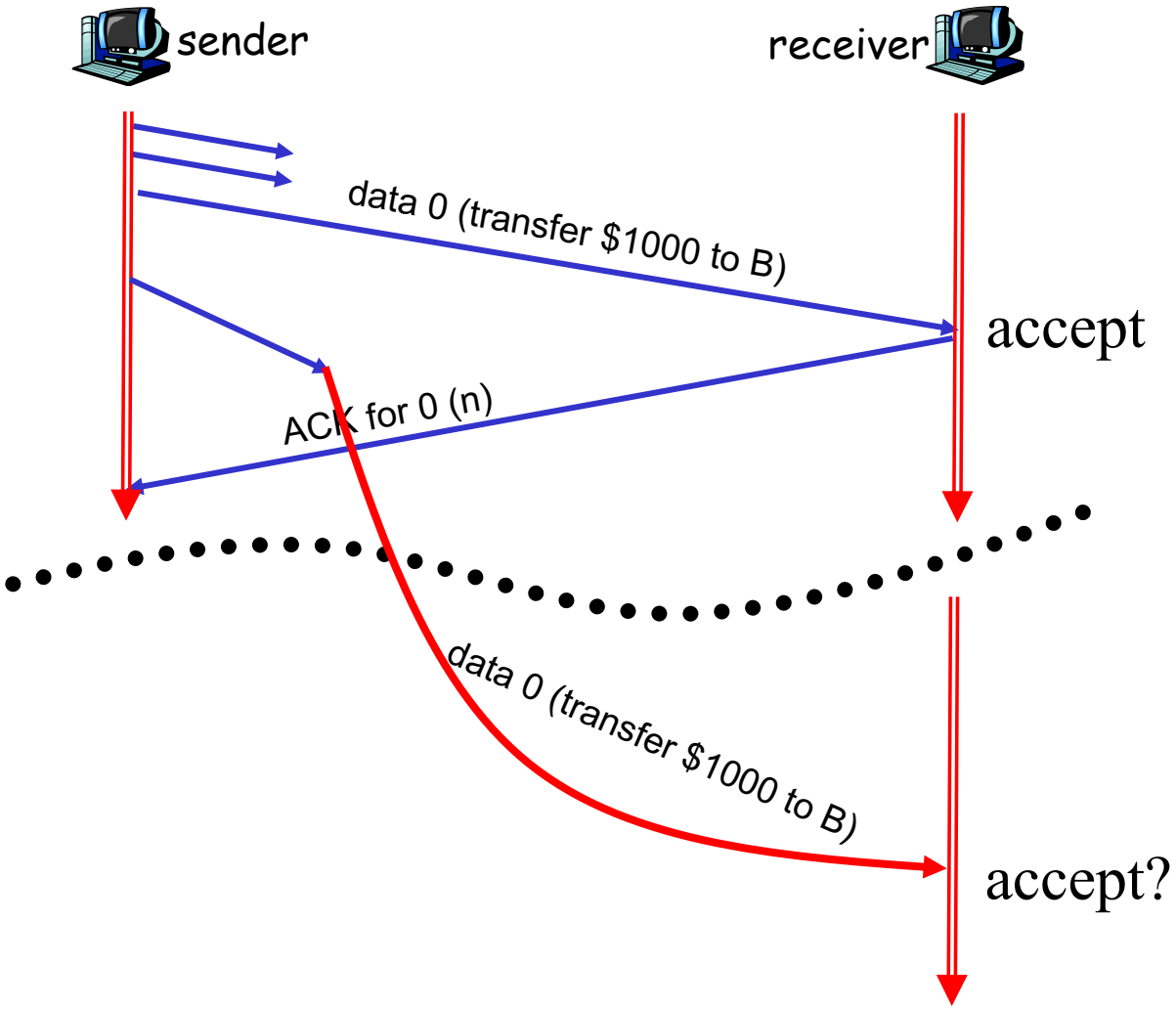
Outline

- Admin and Recap
- TCP reliability
  - data seq#, ack, buffering
  - timeout realization
  - *connection management*

Why Connection Setup/When to Accept (Safely Deliver) First Packet?



Why Connection Setup/When to Accept (Safely Deliver) First Packet?



Recap: Transport "Safe-Setup" Principle

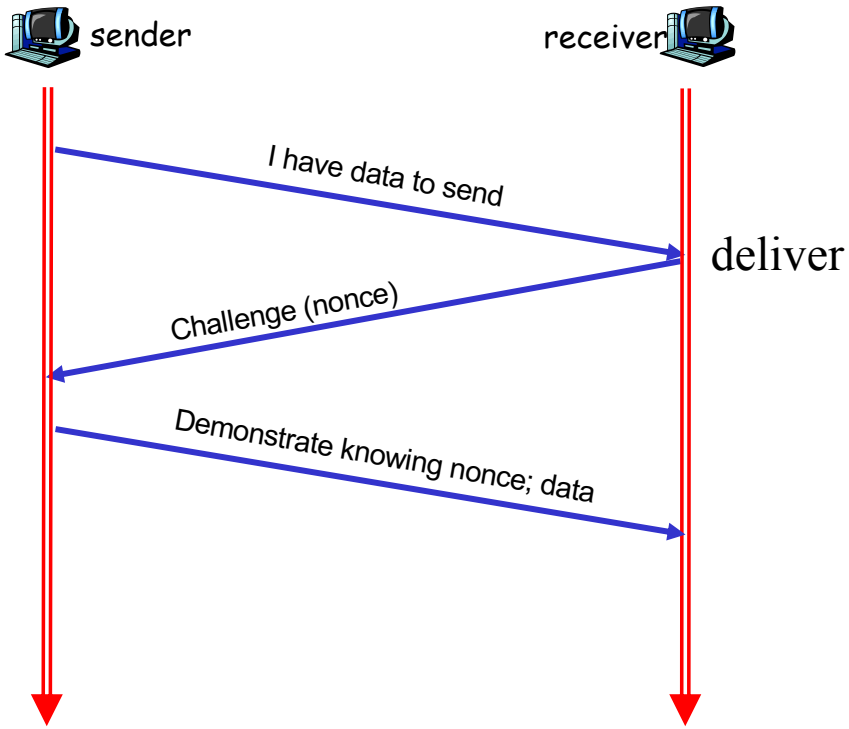
- A general safety principle for a receiver R to accept a message from a sender S is the general "authentication" principle, which consists of two conditions:

Transport authentication principle:
- [p1] Receiver can be sure that what Sender says is **fresh**
- [p2] Receiver receives something that ***only*** Sender can say

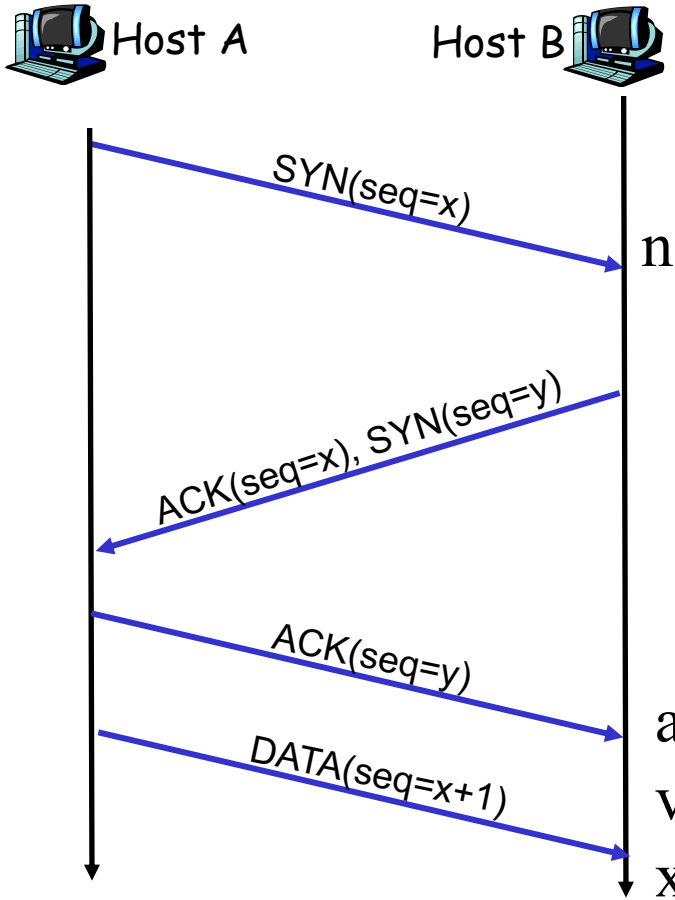
We first assume a secure setting: no malicious attacks.

Exercise: Techniques to allow a receiver to check for freshness (e.g., add a time stamp)?

Generic Challenge-Response Structure Checking Freshness



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

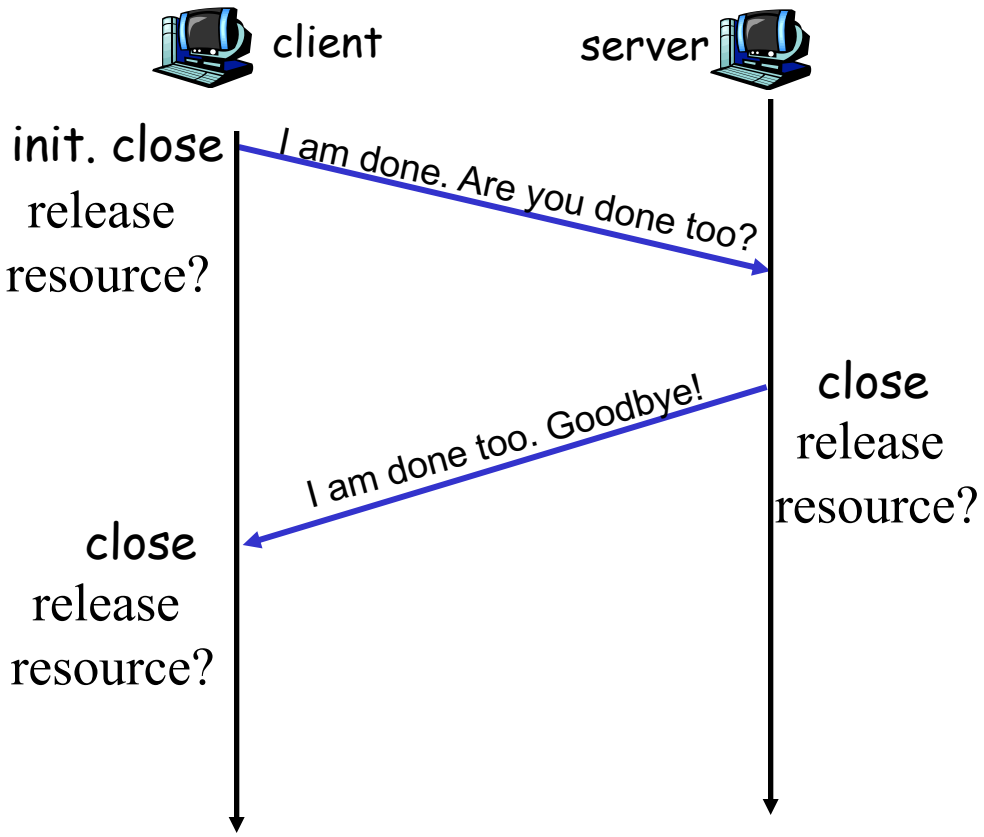
SYN: indicates connection setup

Make "Challenge y" Robust

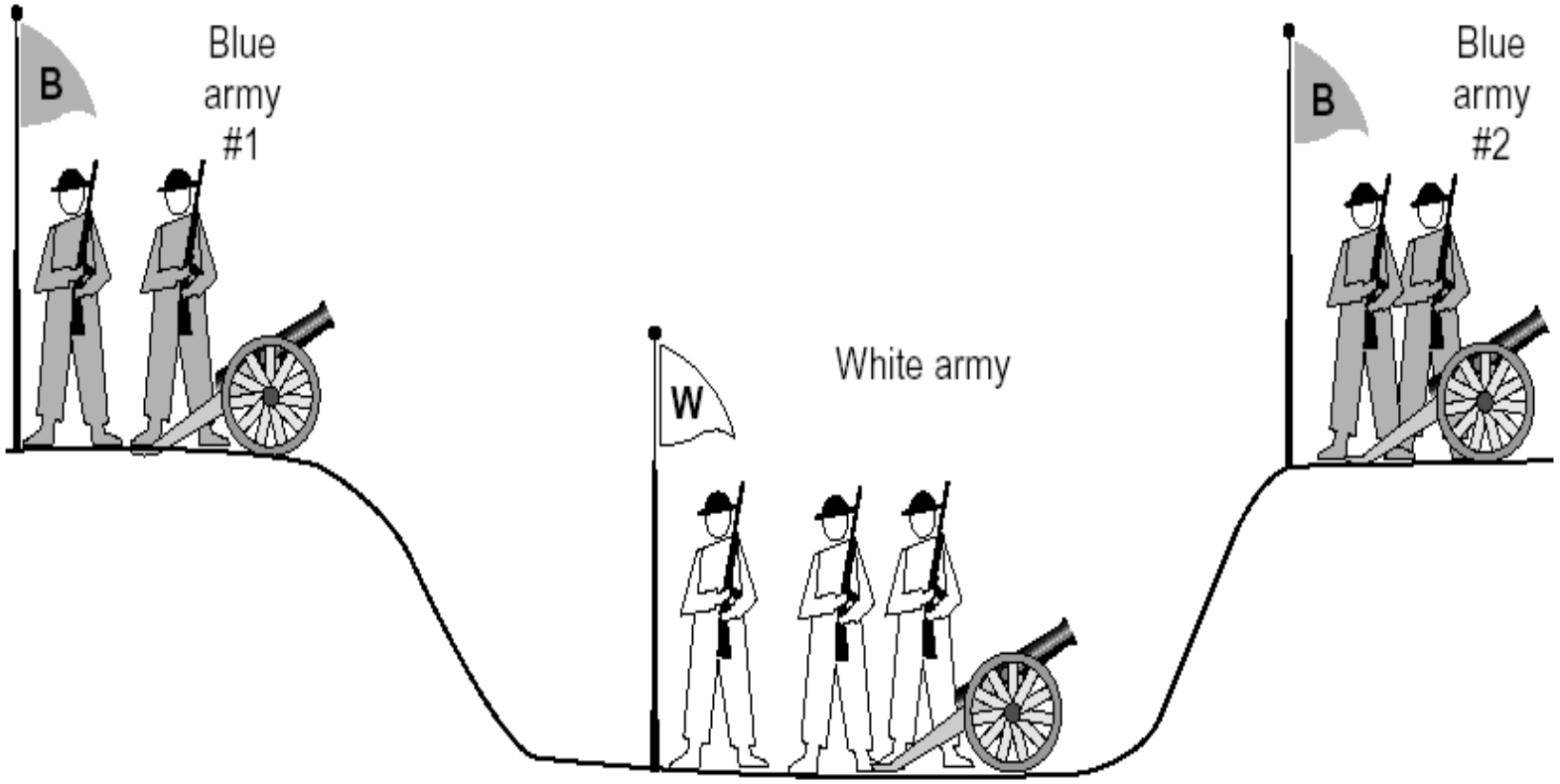
- To avoid that "SYNC ACK y" comes from reordering and duplication
  - for each connection (sender-receiver pair), ensuring that two identically numbered packets are never outstanding at the same time
    - network bounds the life time of each packet
    - a sender will not reuse a seq# before it is sure that all packets with the seq# are purged from the network
    - seq. number space should be large enough to not limit transmission rate
- Increasingly move to cryptographic challenge and response

Connection Close

- Why connection close?
  - so that each side can release resource and remove state about the connection (do not want dangling socket)



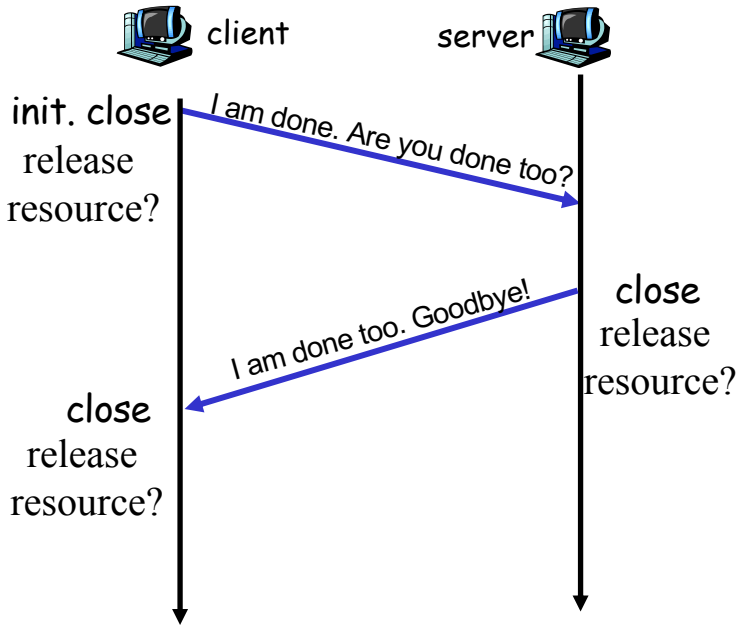
General Case: The Two-Army Problem

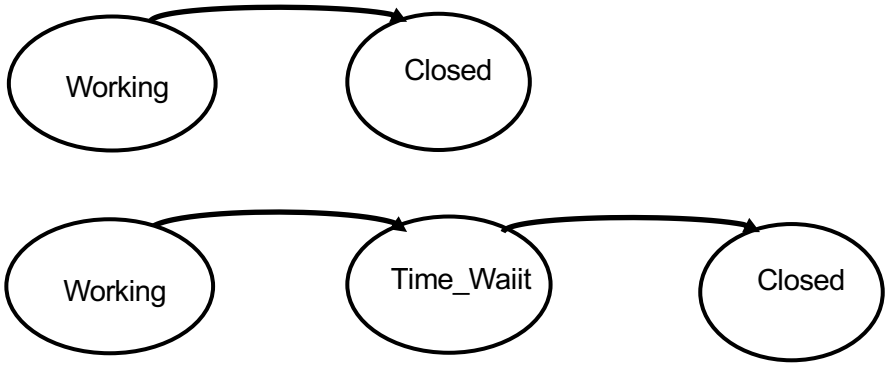


The gray (blue) armies need to agree on whether or not they will attack the white army. They achieve agreement by sending messengers to the other side. If they both agree, attack; otherwise, no. Note that a messenger can be captured!

Time_Wait

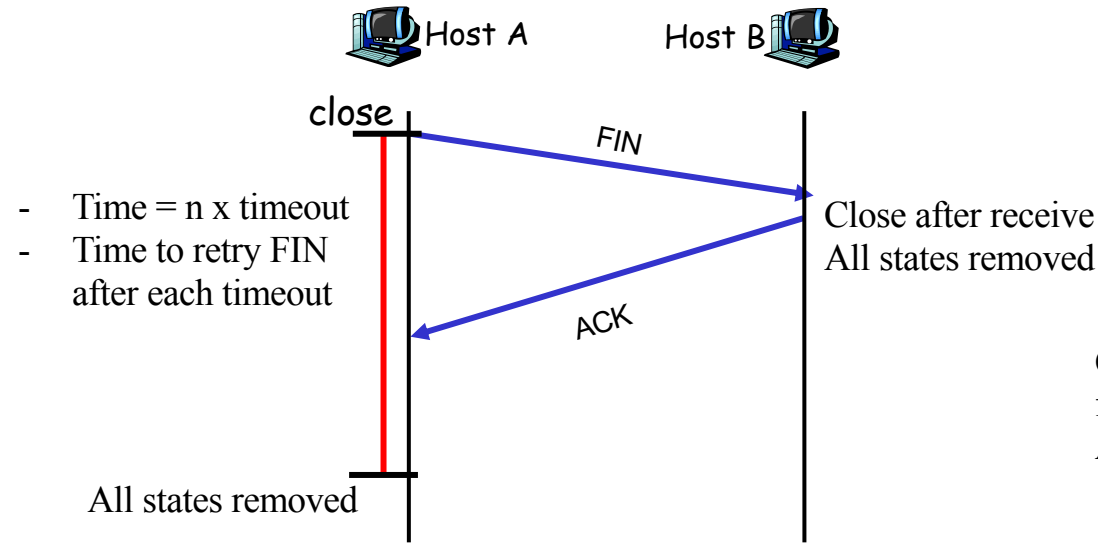
- Generic technique: Timeout to "solve" infeasible problem
  - Instead of message-driven state transition, use a timeout based transition; use timeout to handle error cases





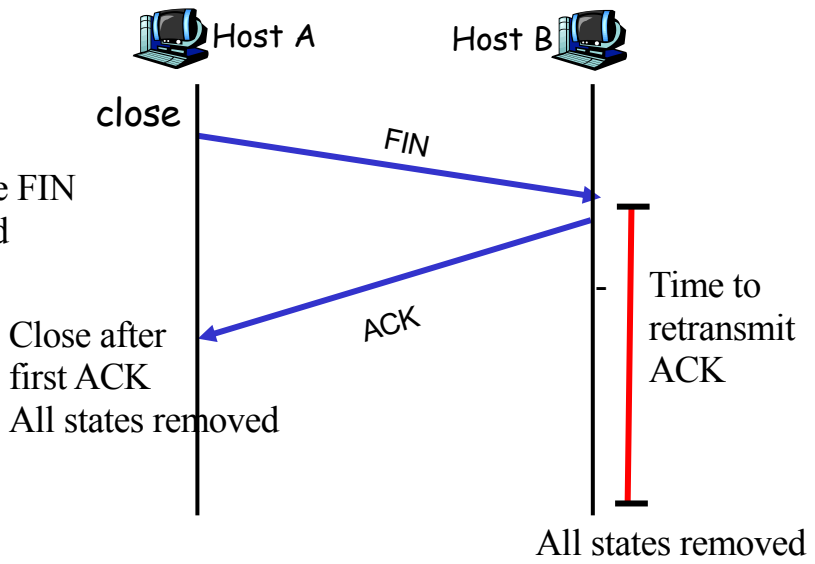
Time_Wait Design Options

Design 1 (initiator time wait)

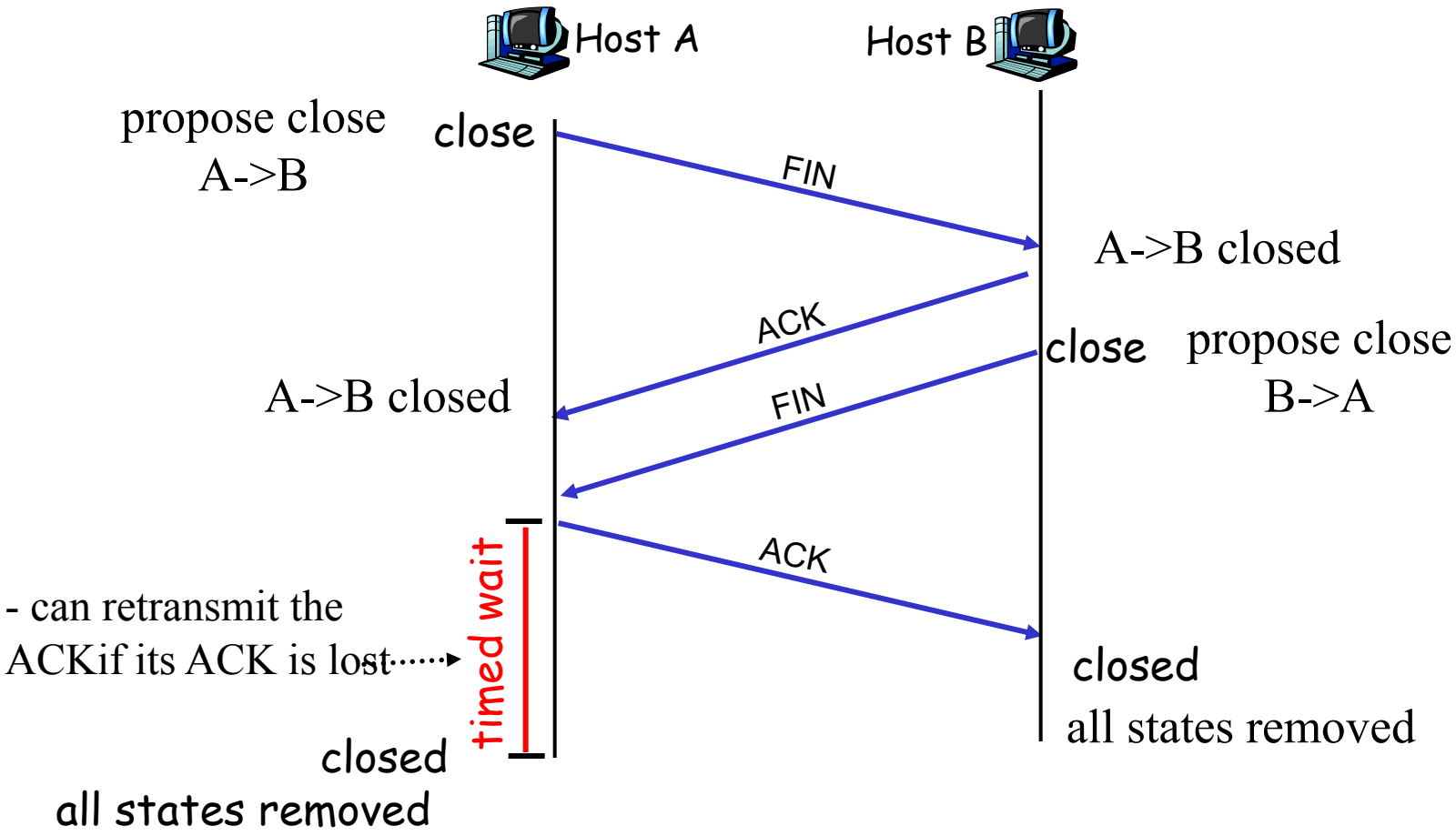


- Time = n x timeout
- Time to retry FIN after each timeout

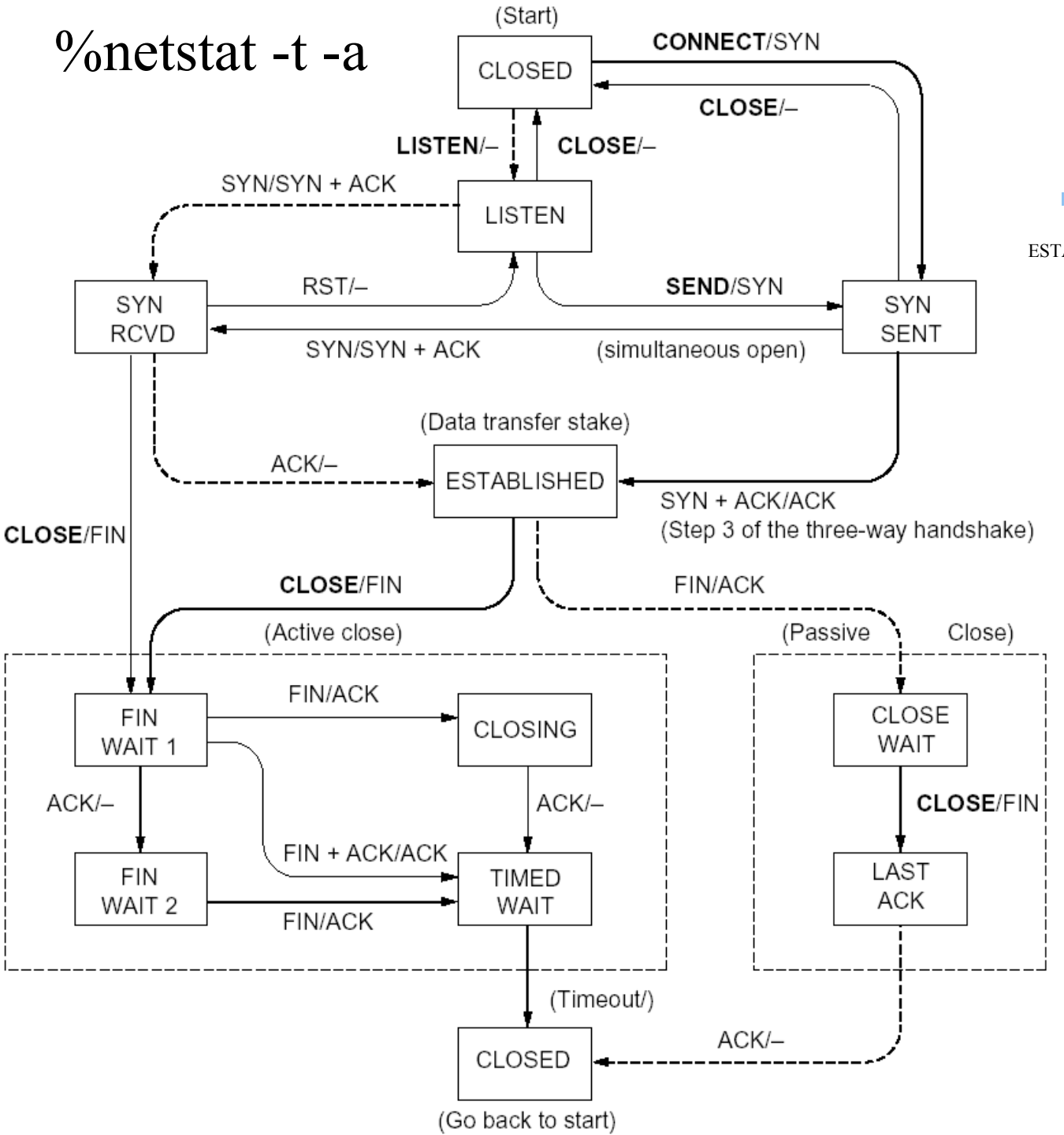
Design 2 (receiver time wait)

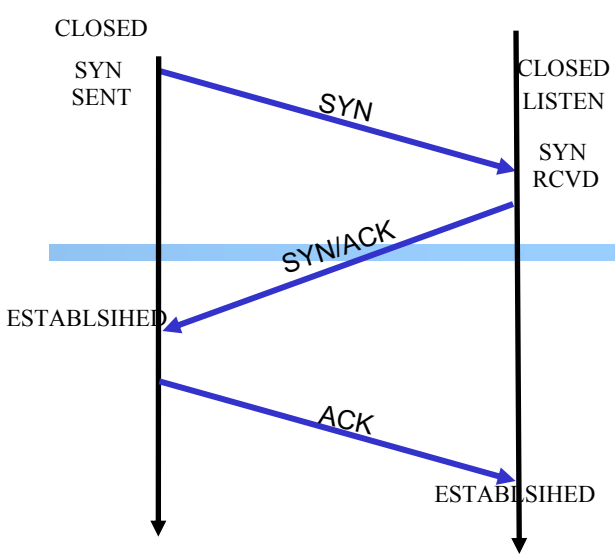


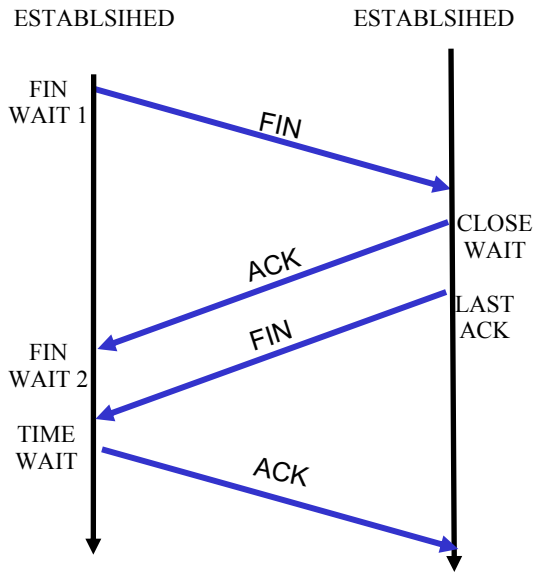
TCP Four Way Teardown (For Bi-Directional Transport)



%netstat -t -a

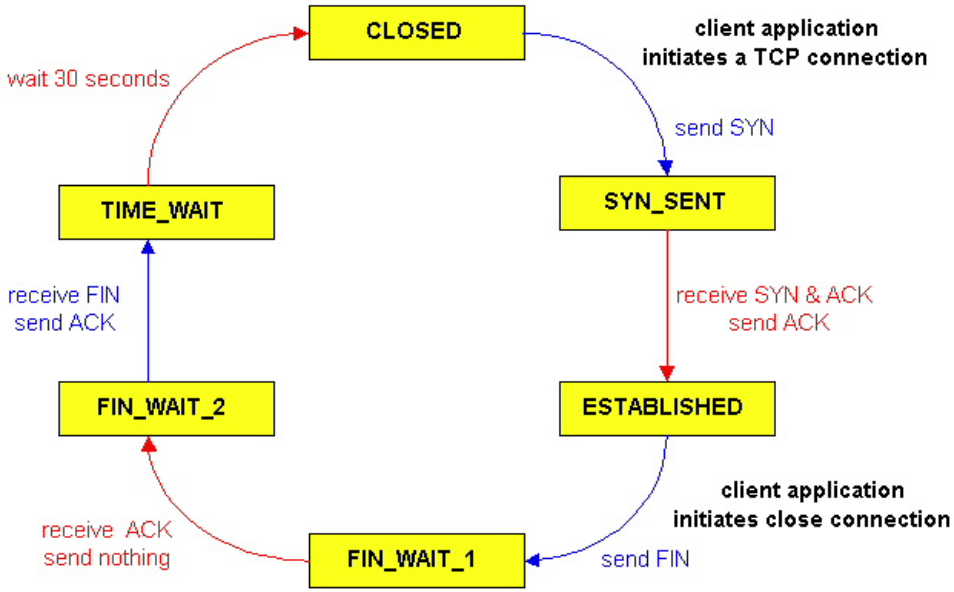

(Start)
CLOSED
CONNECT/SYN
CLOSE/–
LISTEN/–
CLOSE/–
SYN/SYN + ACK
LISTEN
SYN RCVD
RST/–
SEND/SYN
SYN SENT
SYN/SYN + ACK
(simultaneous open)
(Data transfer stake)
ACK/–
ESTABLISHED
SYN + ACK/ACK
(Step 3 of the three-way handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
(Active close)
(Passive Close)
FIN WAIT 1
FIN/ACK
CLOSING
CLOSE WAIT
ACK/–
ACK/–
CLOSE/FIN
FIN + ACK/ACK
FIN WAIT 2
TIMED WAIT
LAST ACK
FIN/ACK
(Timeout/)
CLOSED
ACK/–
(Go back to start)

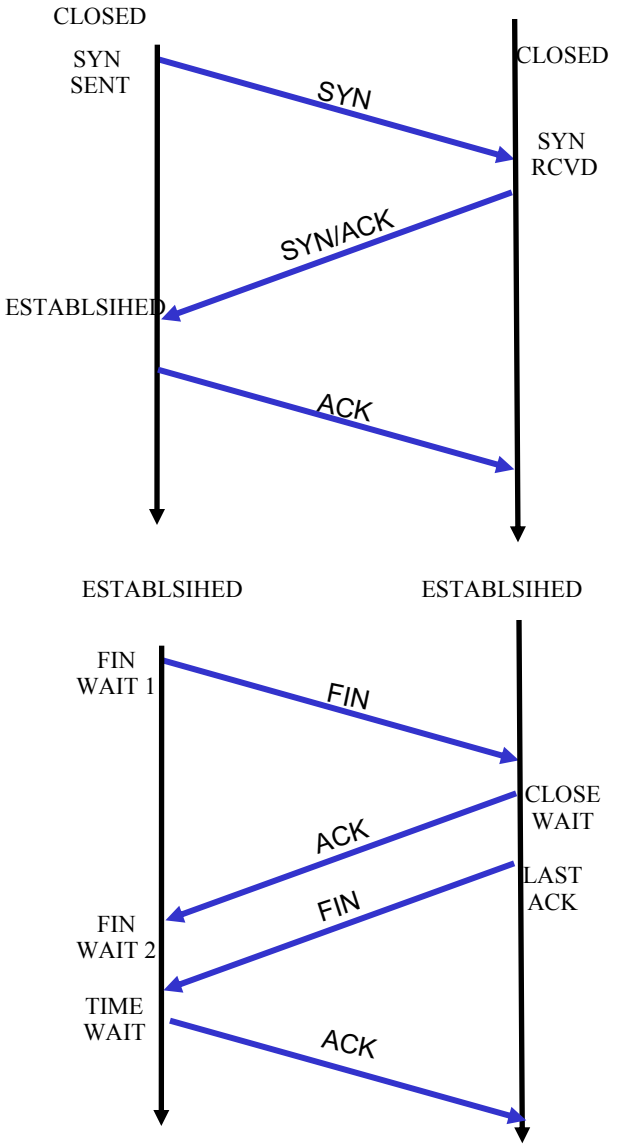

CLOSED
SYN SENT
SYN
CLOSED LISTEN
SYN RCVD
SYN/ACK
ESTABLSIHED
ACK
ESTABLSIHED


ESTABLSIHED
ESTABLSIHED
FIN WAIT 1
FIN
CLOSE WAIT
ACK
LAST ACK
FIN
FIN WAIT 2
TIME WAIT
ACK

TCP Connection Management

TCP lifecycle: init SYN/FIN

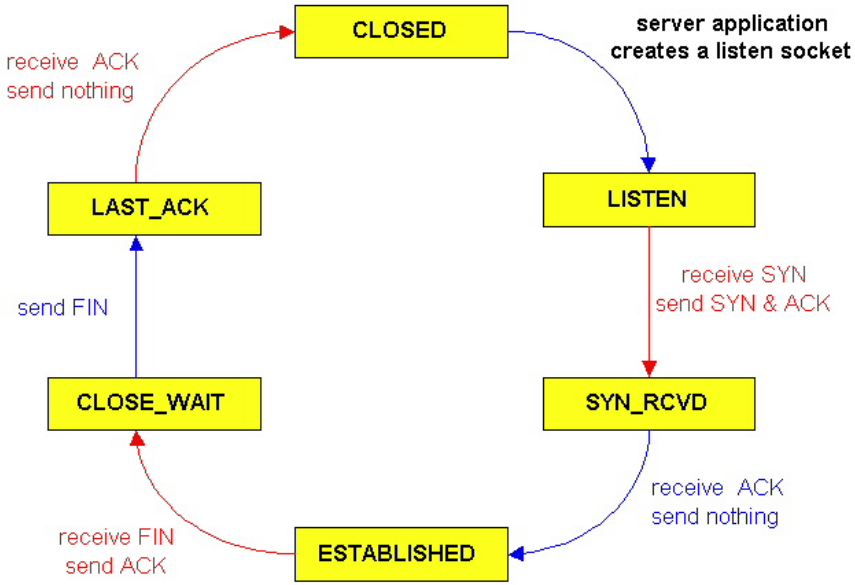


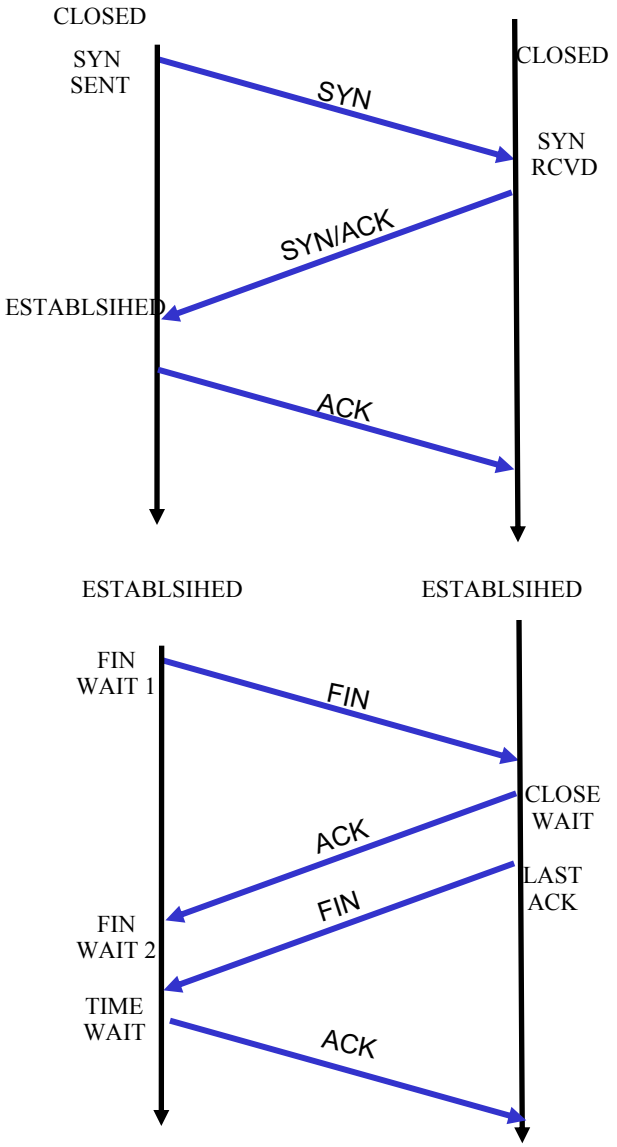


http://dsd.lbl.gov/TCP-tuning/ip-sysctl-2.6.txt

TCP Connection Management

TCP lifecycle: wait for SYN/FIN



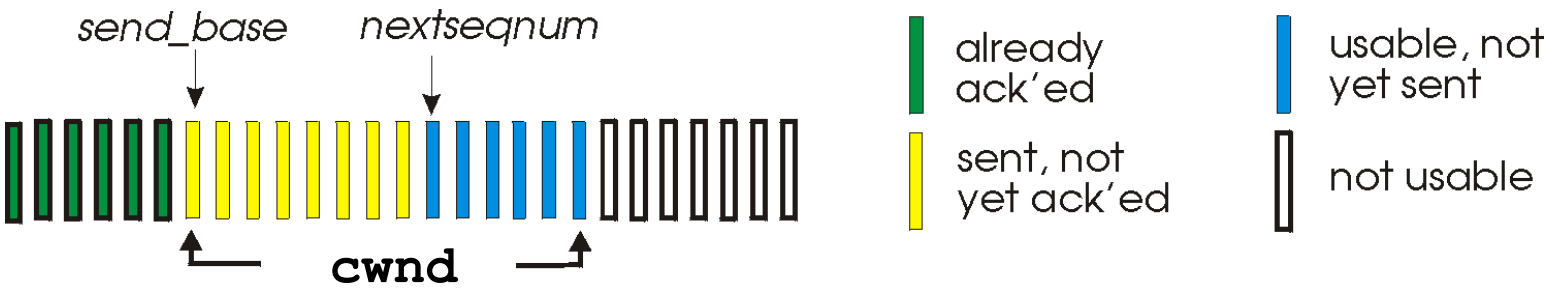


A Summary of Questions

- Basic structure: sliding window protocols
- How to 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?

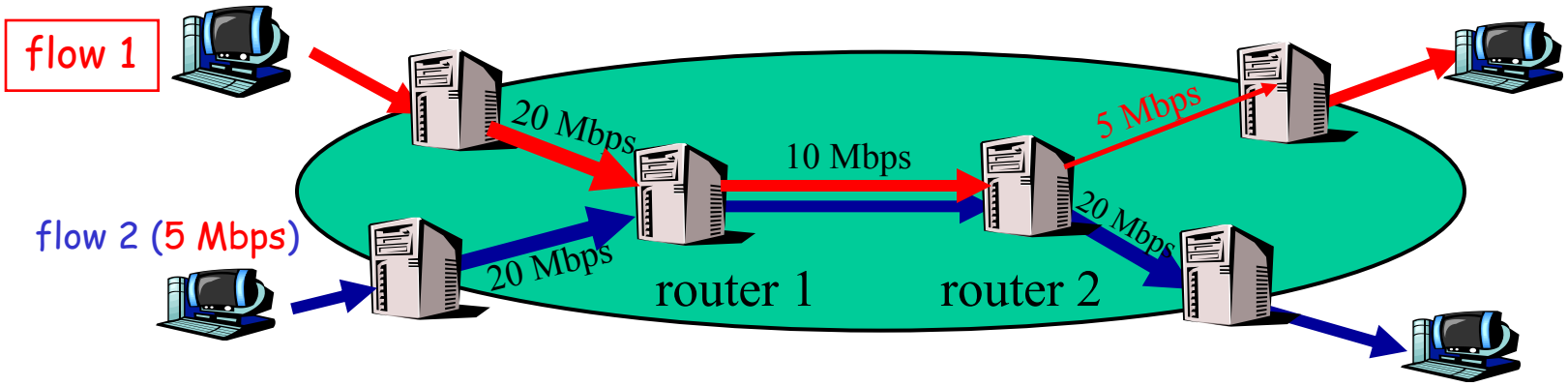
Roadmap

- 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

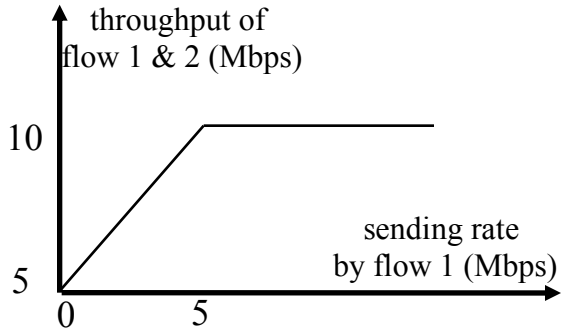
- Admin and recap
- TCP Reliability
- Transport congestion control
  - *what is congestion (cost of congestion)*

Cause/Cost of Congestion: Single Bottleneck

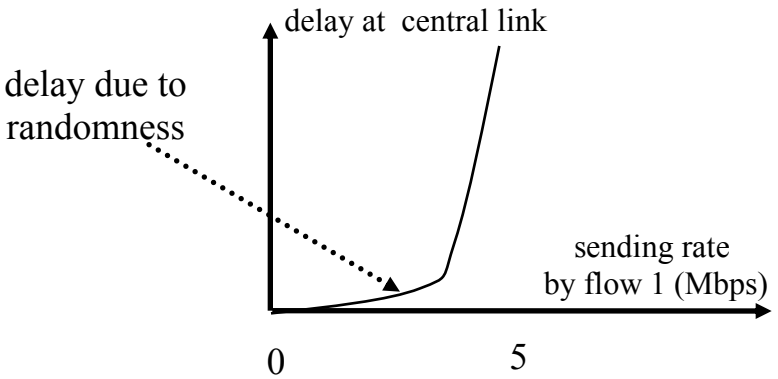


- Flow 2 has a fixed sending rate of 5 Mbps
- We vary the sending rate of flow 1 from 0 to 20 Mbps
- Assume
  - no retransmission; link from router 1 to router 2 has infinite buffer

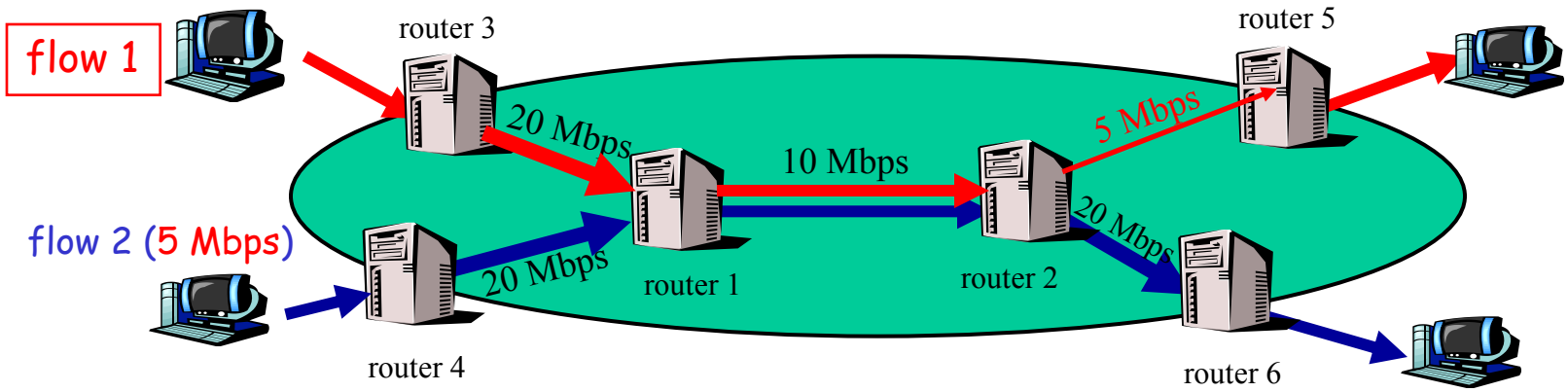
throughput: e2e packets delivered in unit time



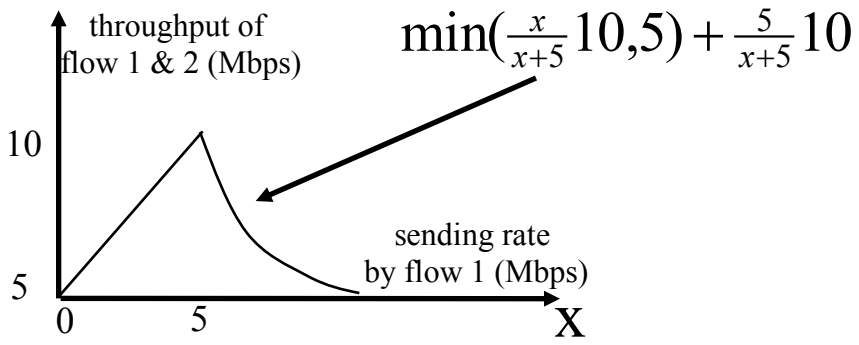
Delay?



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

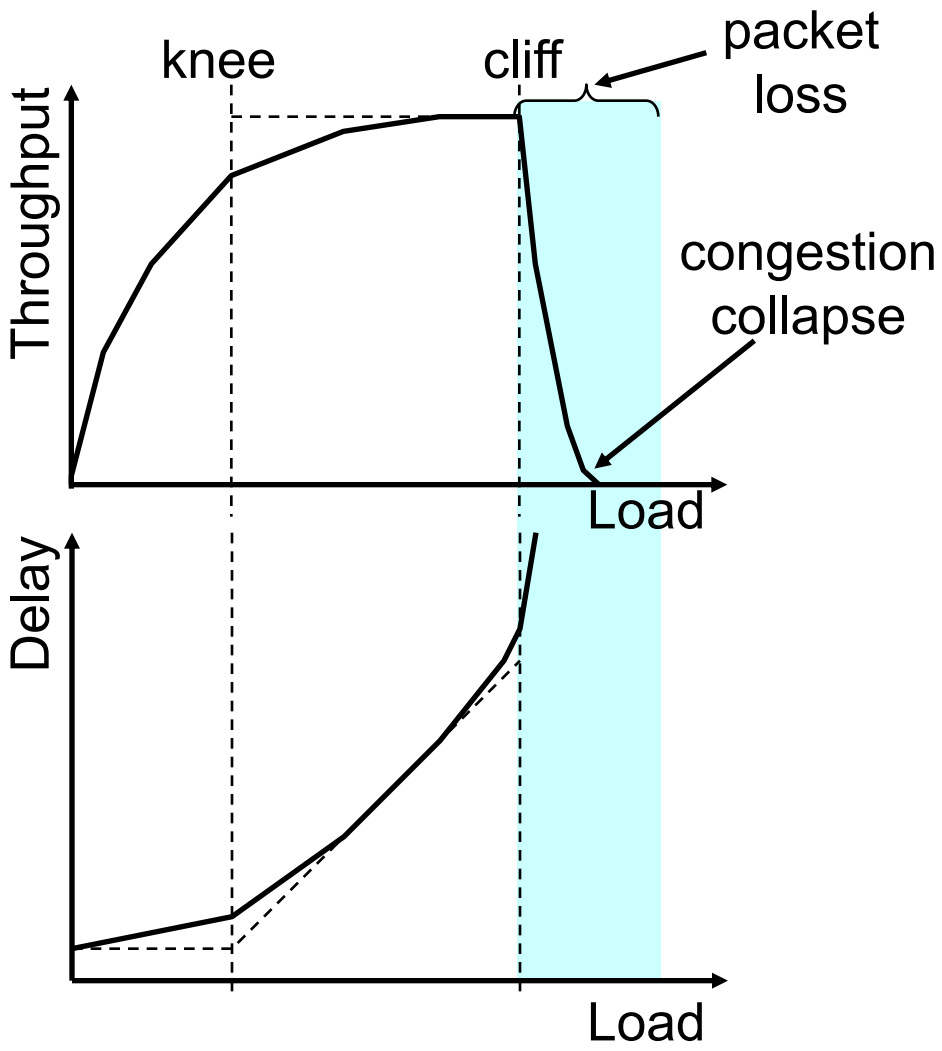


- Zombie packet: a packet dropped at the link from router 2 to router 5; the upstream transmission from router 1 to router 2 used for that packet was wasted!

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

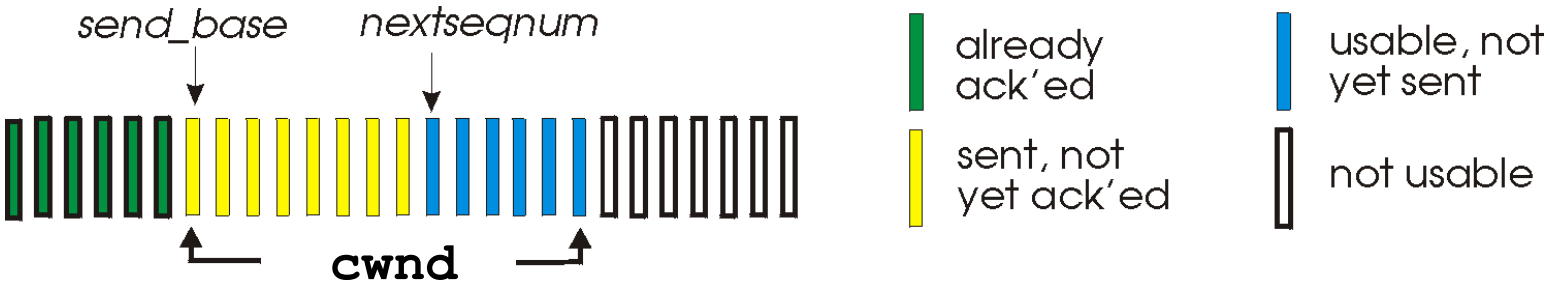
Window-based:

- 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:

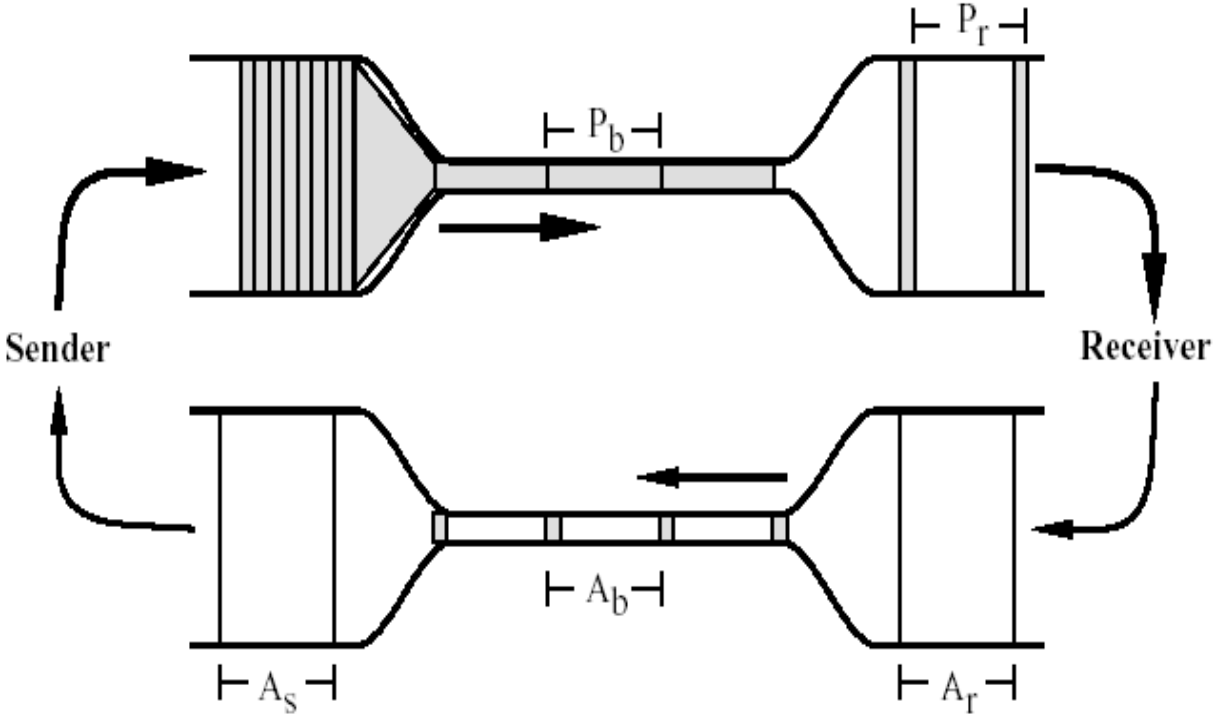


- cwnd segments, each with MSS bytes sent in one RTT:

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

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

Window-based Congestion Control

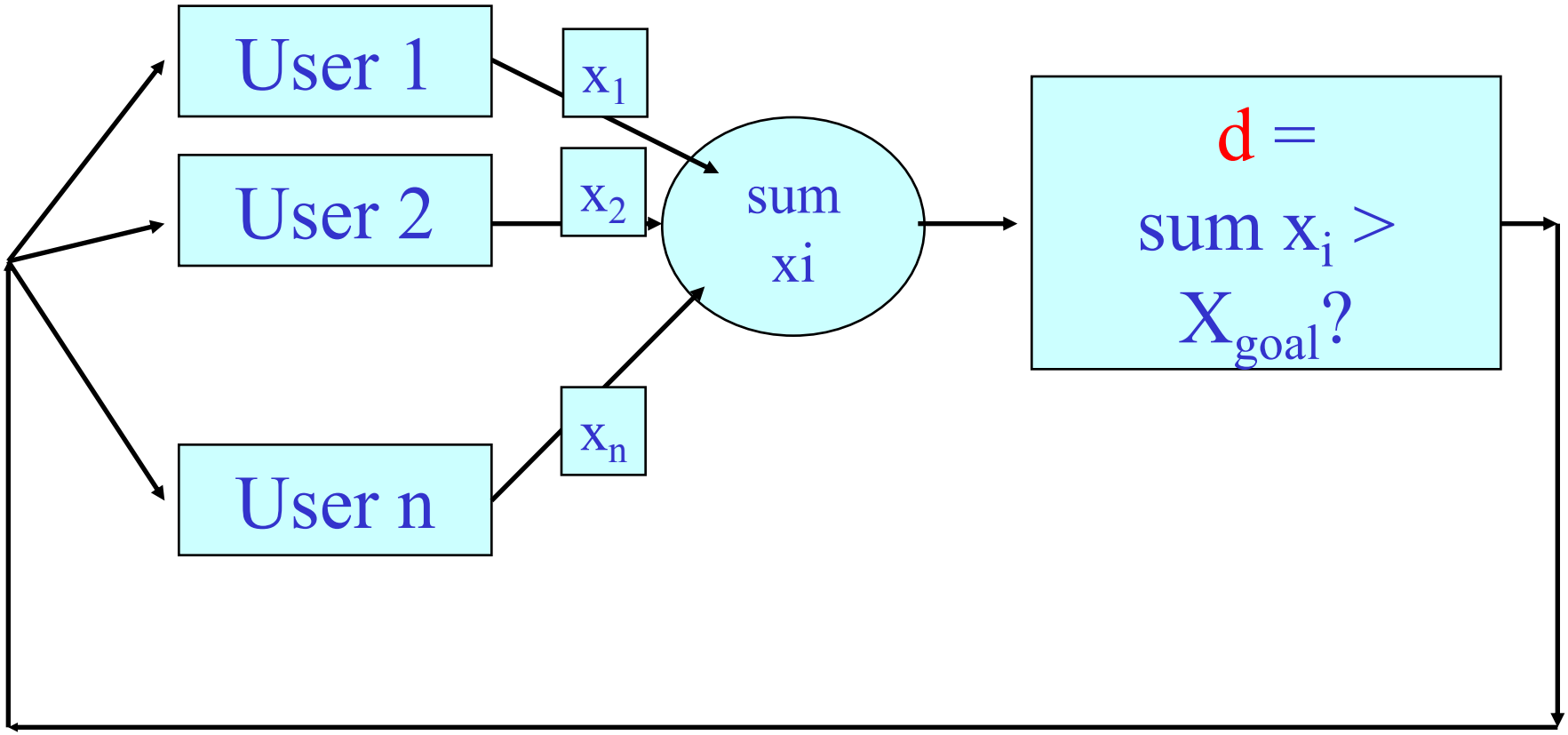


- Window-based congestion control is self-clocking: considers flow conservation, and adjusts to RTT variation automatically.
- Hence, for better safety, more designs use window-based design.

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

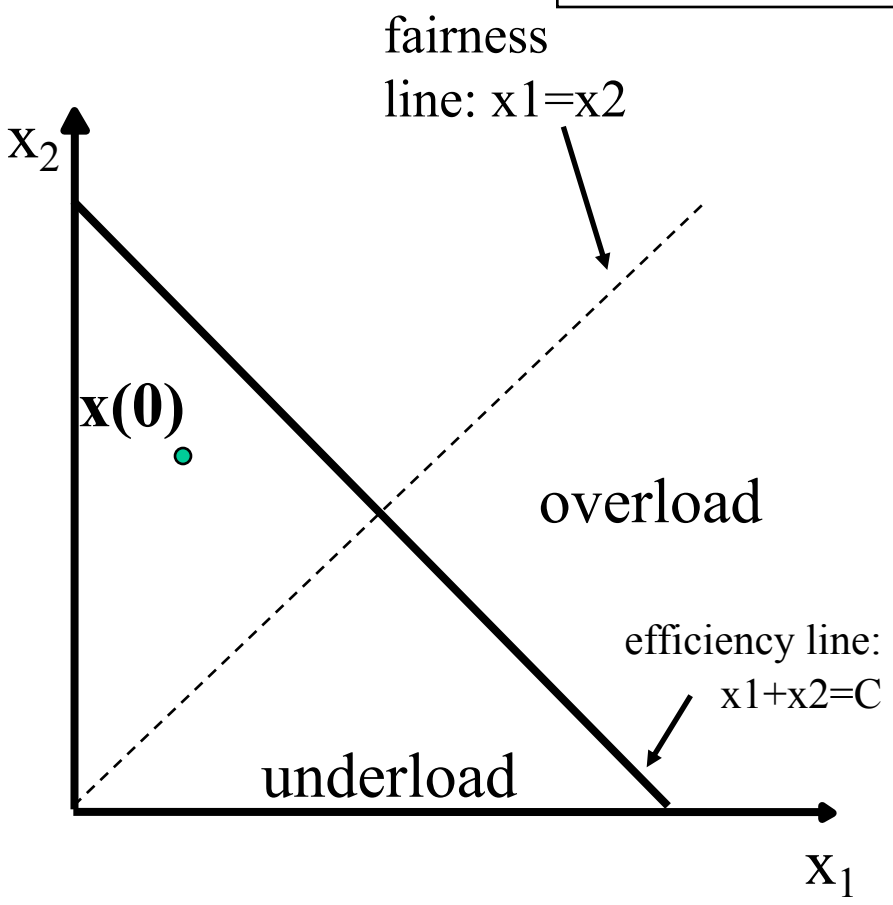
- Proposed by Chiu and Jain (1988)
- The simplest control strategy

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

Discussion: values of the parameters?

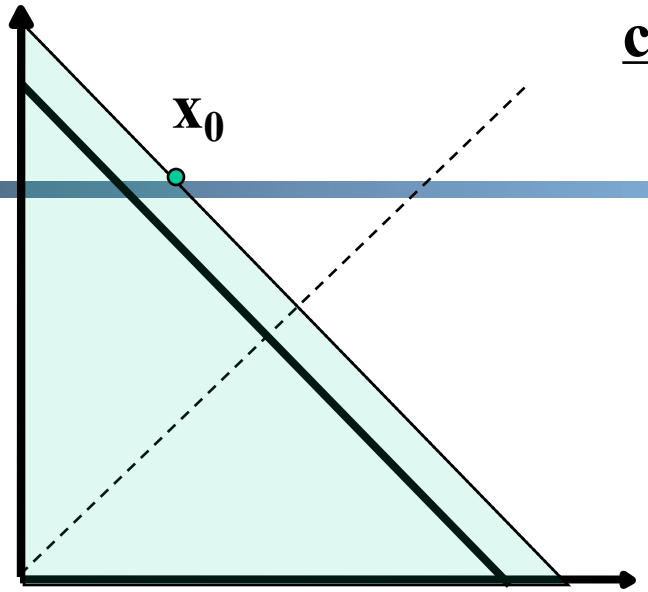
State Space of Two Flows

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

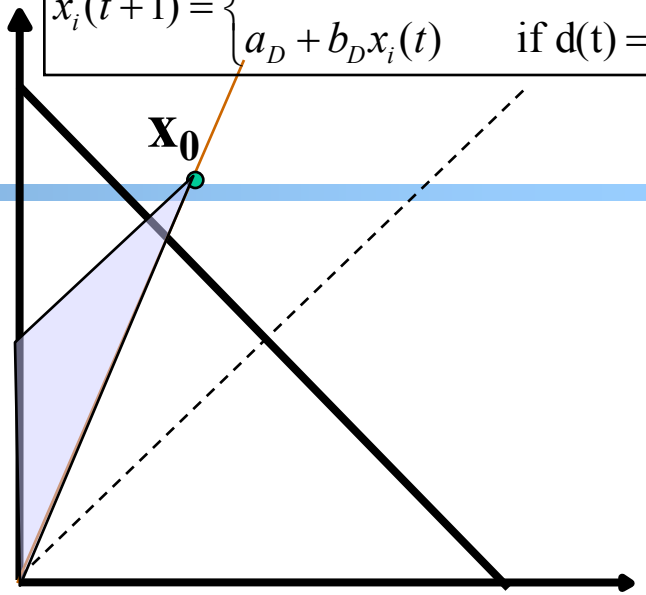


congestion

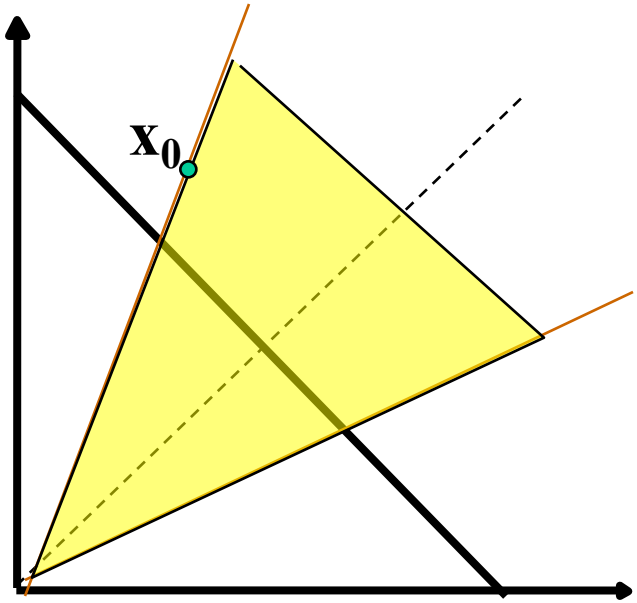
$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{if d(t) = no cong.} \\ a_D + b_D x_i(t) & \text{if d(t) = cong.} \end{cases}$$



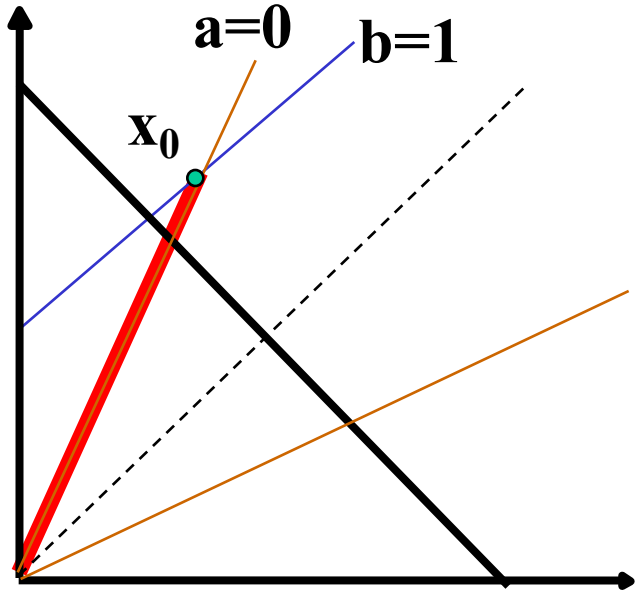
efficiency



efficiency: distributed linear rule



fairness



intersection

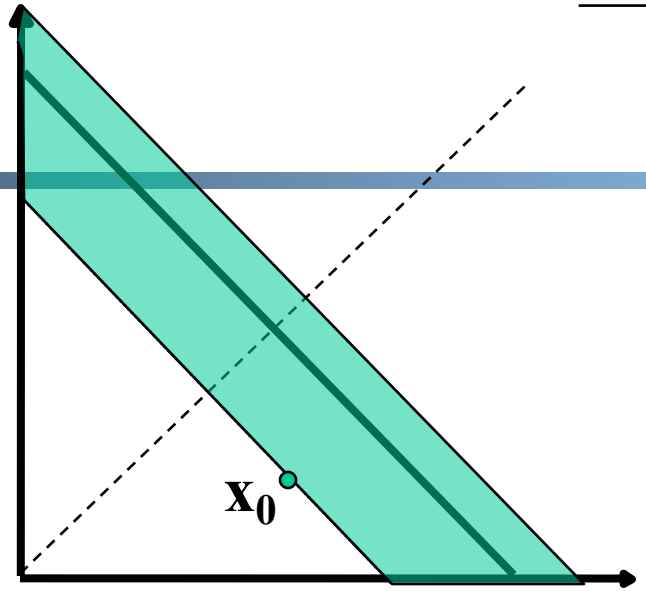
Implication: Congestion (overload) Case

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

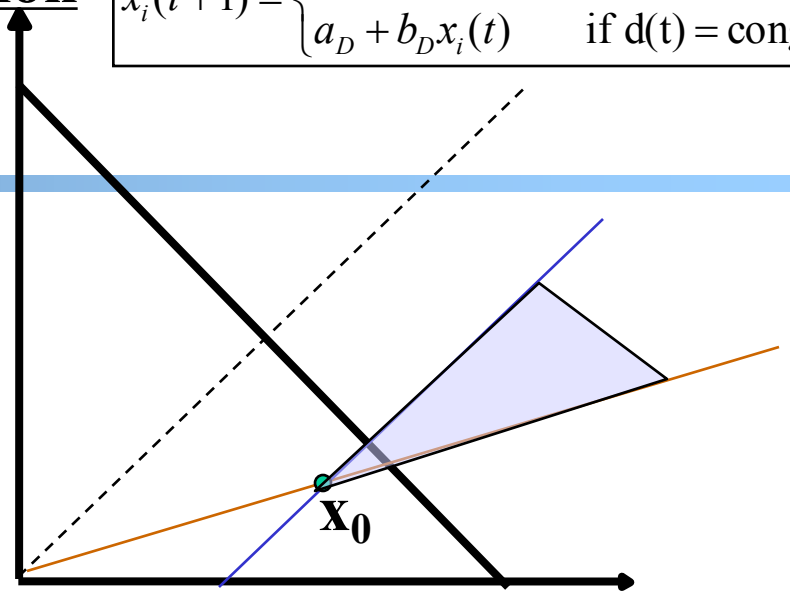
$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{if d(t) = no cong.} \\ b_D x_i(t) & \text{if d(t) = cong.} \end{cases}$$

no-congestion

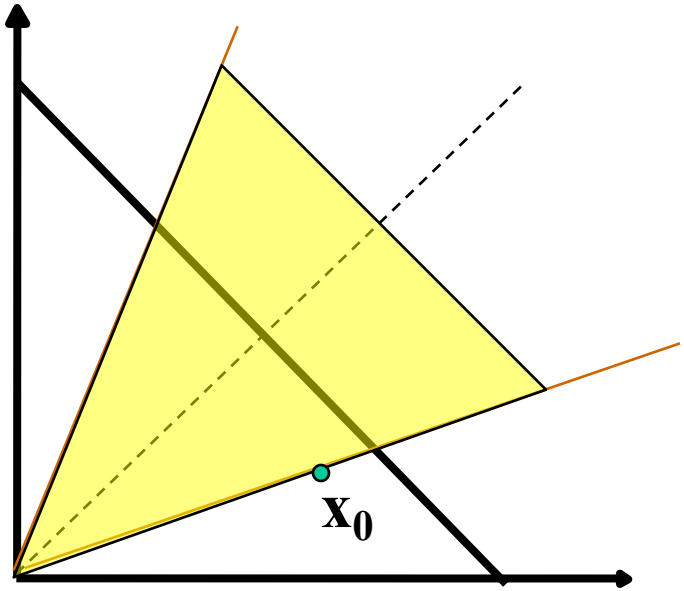
$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{if d(t) = no cong.} \\ a_D + b_D x_i(t) & \text{if d(t) = cong.} \end{cases}$$



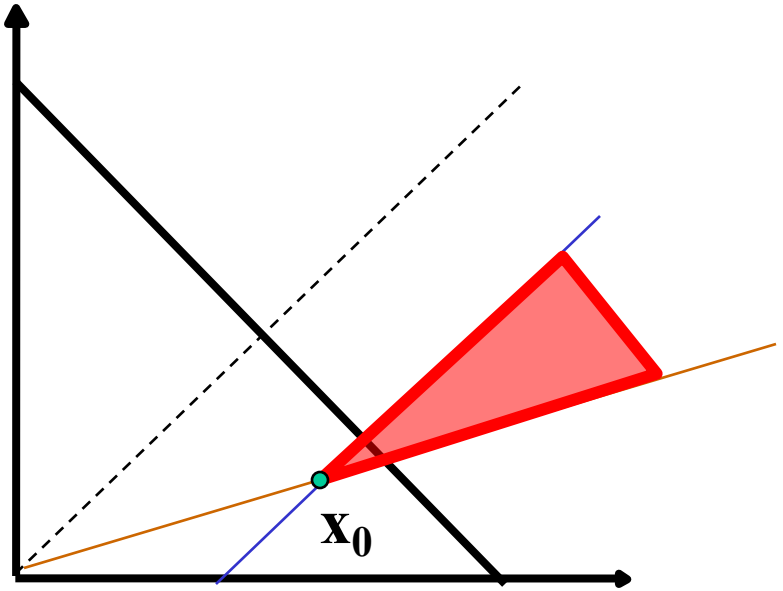
efficiency



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.,
  - $a_I > 0$, $b_I > 1$

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

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

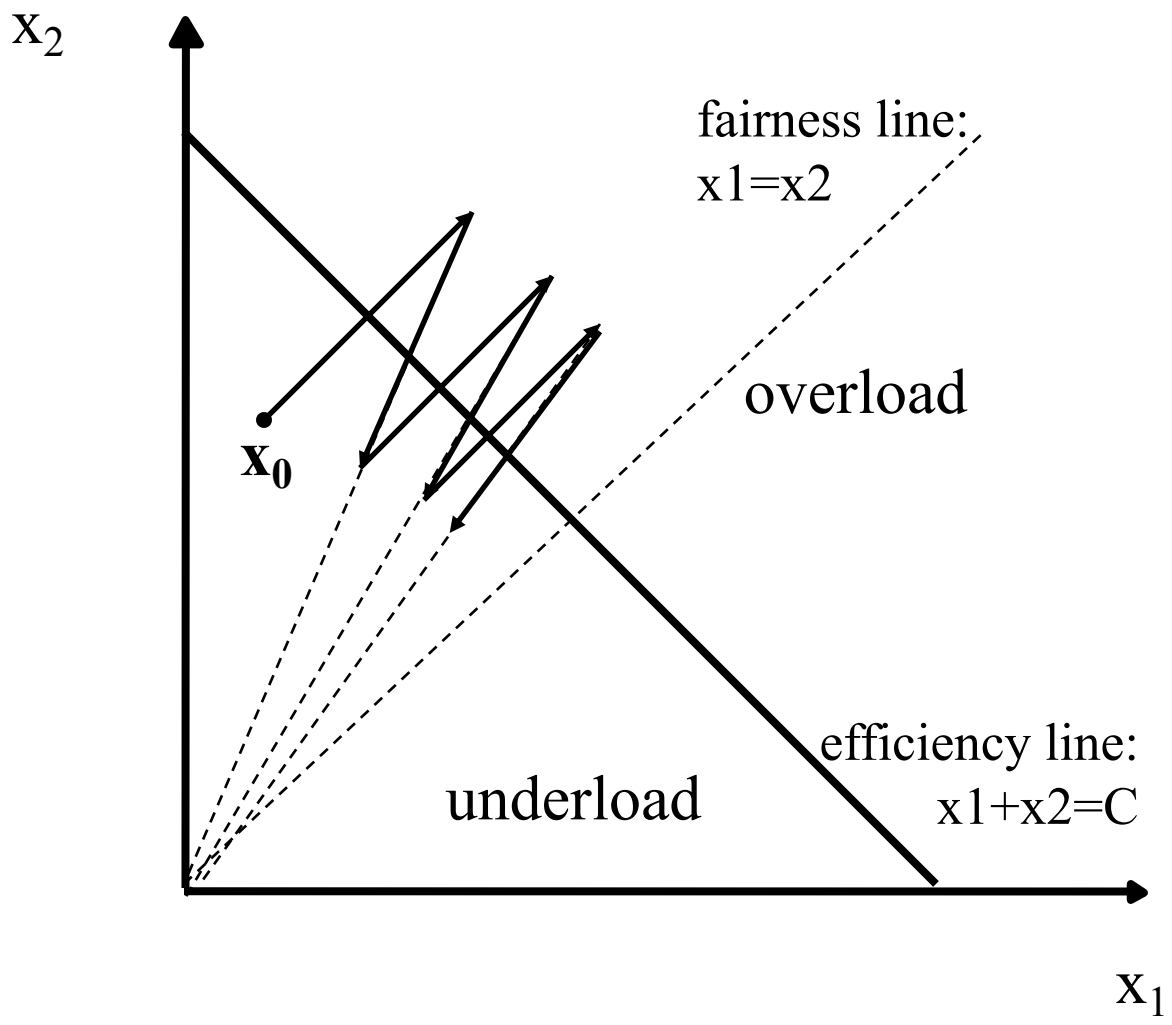
Intuition: State Trace Analysis of Four Special Cases

	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 ($a_I=a_D=0$)

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

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

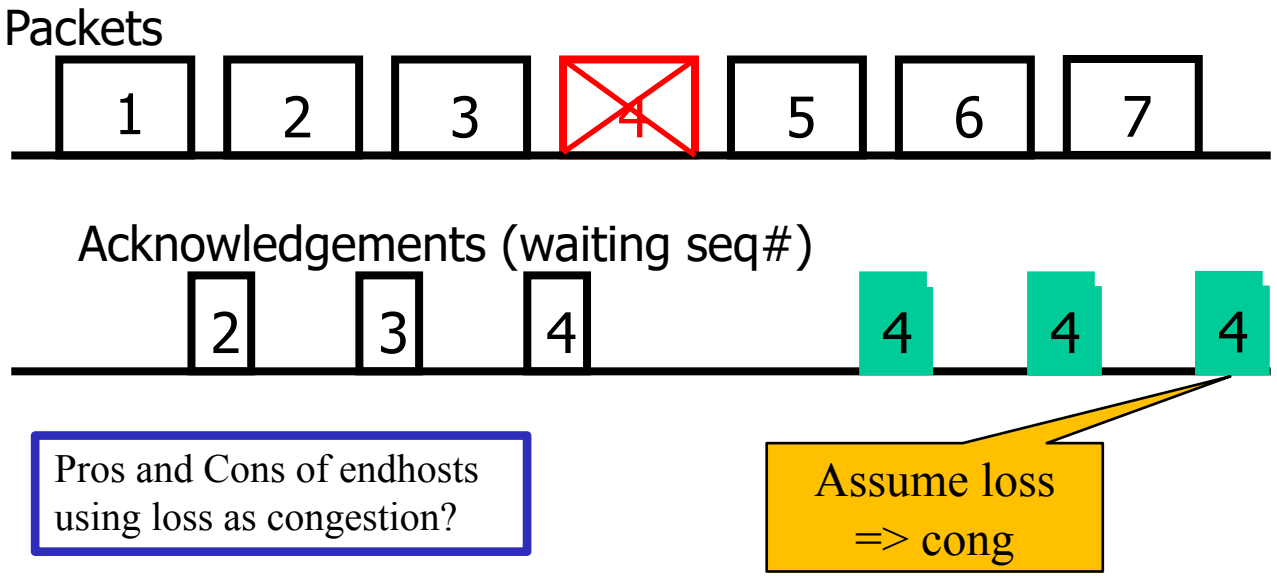
For more details: see TCP/IP illustrated; or read http://lxr.linux.no/source/net/ipv4/tcp_input.c for linux implementation

Mapping A(M)I-MD to Protocol

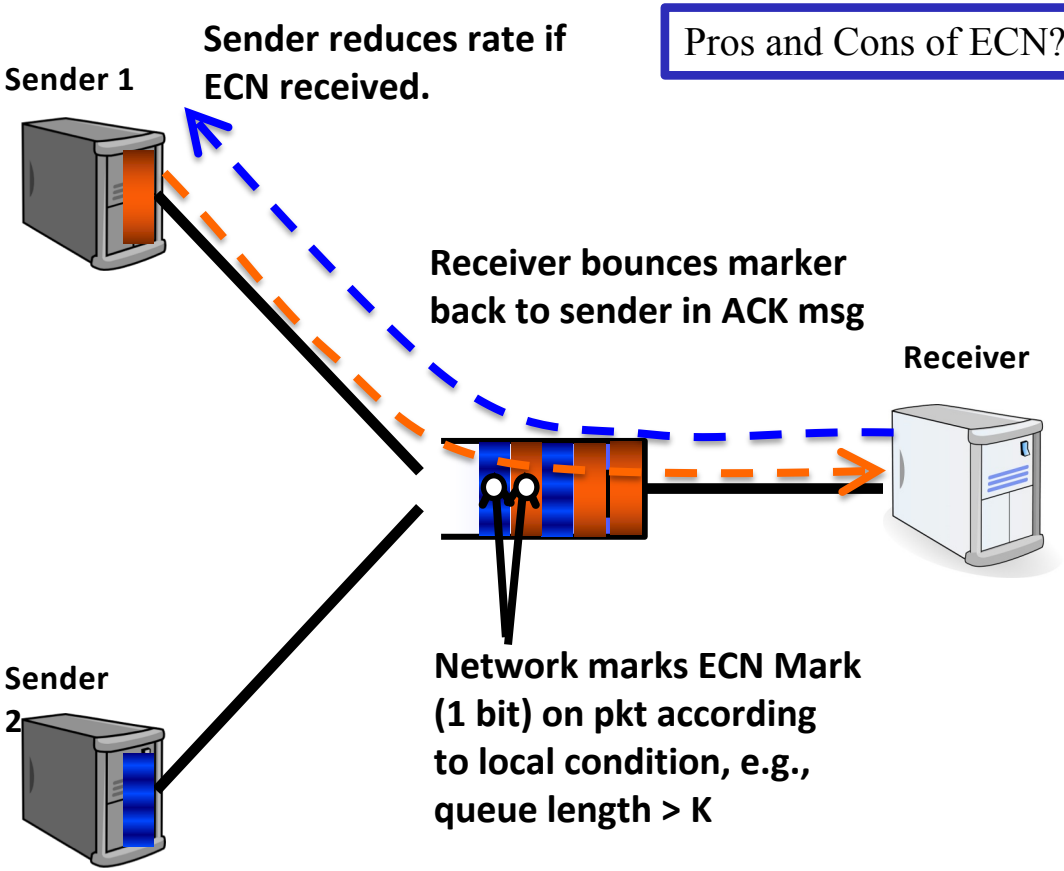
- Basic questions to look at:
  - How to obtain d(t)--the congestion signal?
  - What values do we choose for the formula?
  - How to map formula to code?

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

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



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



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?

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