PADERBORN UNIVERSITY

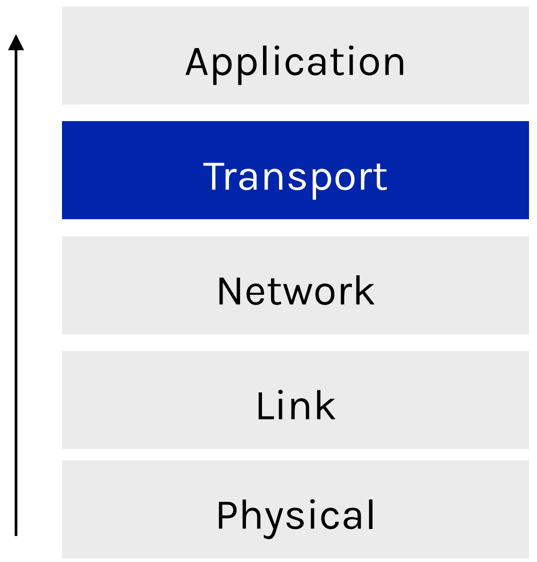
Computer Networks (WS23/24)

L9: The Transport Layer - Part 3

Prof. Dr. Lin Wang
Computer Networks Group (PBNet)
Department of Computer Science
Paderborn University

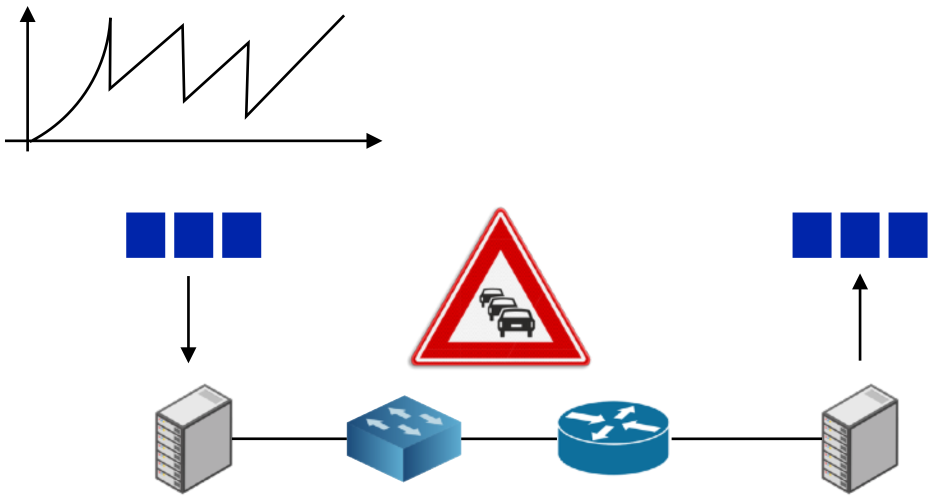
Materials inspired by Laurent Vanbever

Learning objectives



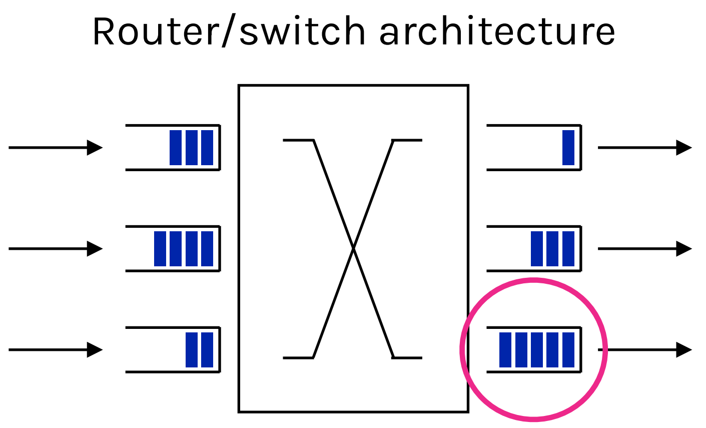
Part 3

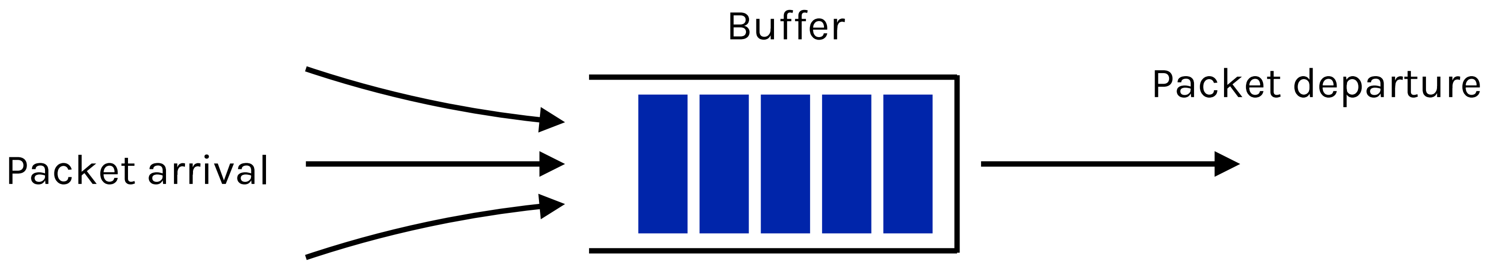
- Network congestion
- Congestion detection and control
- Explicit Congestion Notification (ECN)



Network Congestion

Why congestion happens?





Buffer has a **limited** size; buffer **overflow** if too many packets arrive within a short period of time

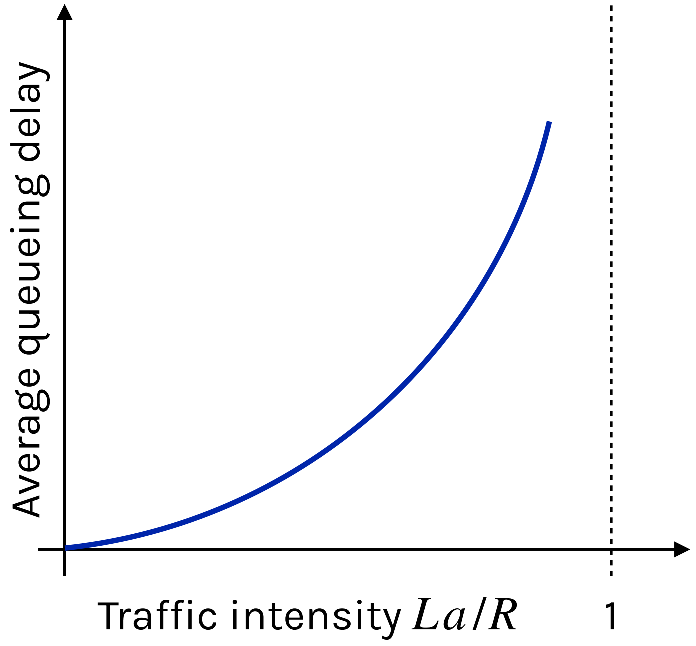
Quantifying congestion

Parameters

- Average packet arrival rate a, transmission rate R, fixed packet length L
- **Traffic intensity** is given by La/R

When the traffic intensity is > 1, the queue will increase without bound, so does the queueing delay

When the traffic intensity is <= 1, queueing delay depends on the burst size



Network congestion is not a new problem

The Internet almost died of congestion in 1986

- Througput collapsed from 32 Kbps to 40 bps

Van Jacobson saved us with Congestion Control

- His solution went right into BSD implementation

Recent resurgence of research interest after brief lag

- New methods (ML-based), new contexts (data centers, 5G), new requirements

Van Jacobson, Michael J. Karels. Congestion Avoidance and Control. ACM SIGCOMM, 1988.

Original network behavior

Upon connection establishment

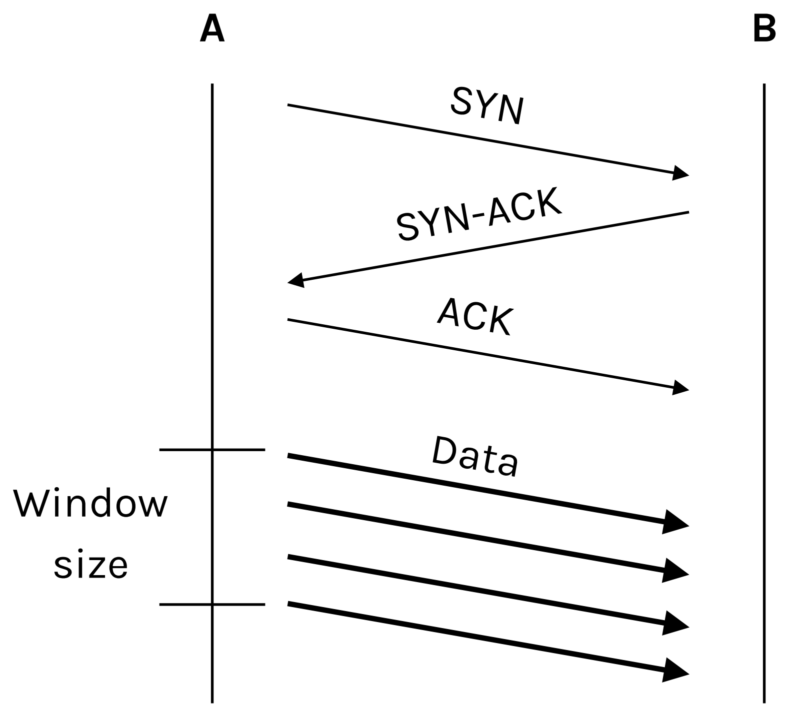
- Send full window of packets

Upon timer expiration

- Retransmit packet immediately

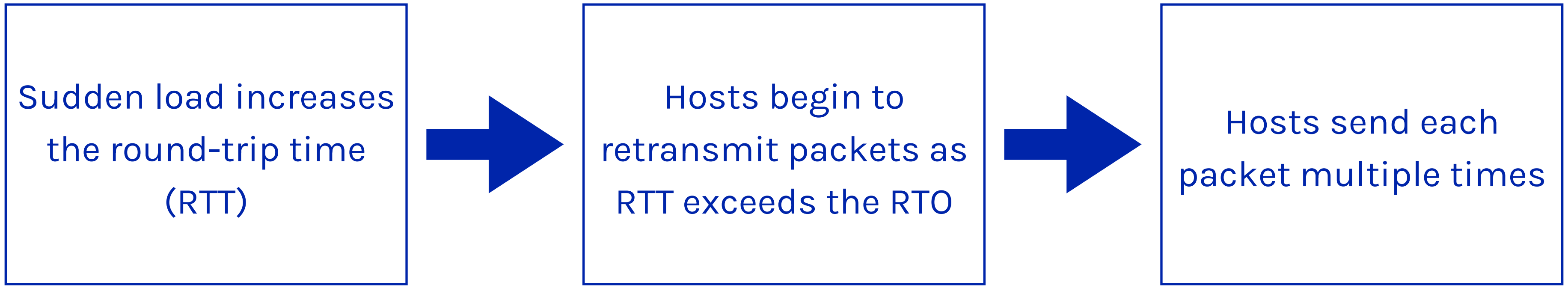
Outcome

- Sending rate only limited by flow control
- Net effect: window-sized burst of packets

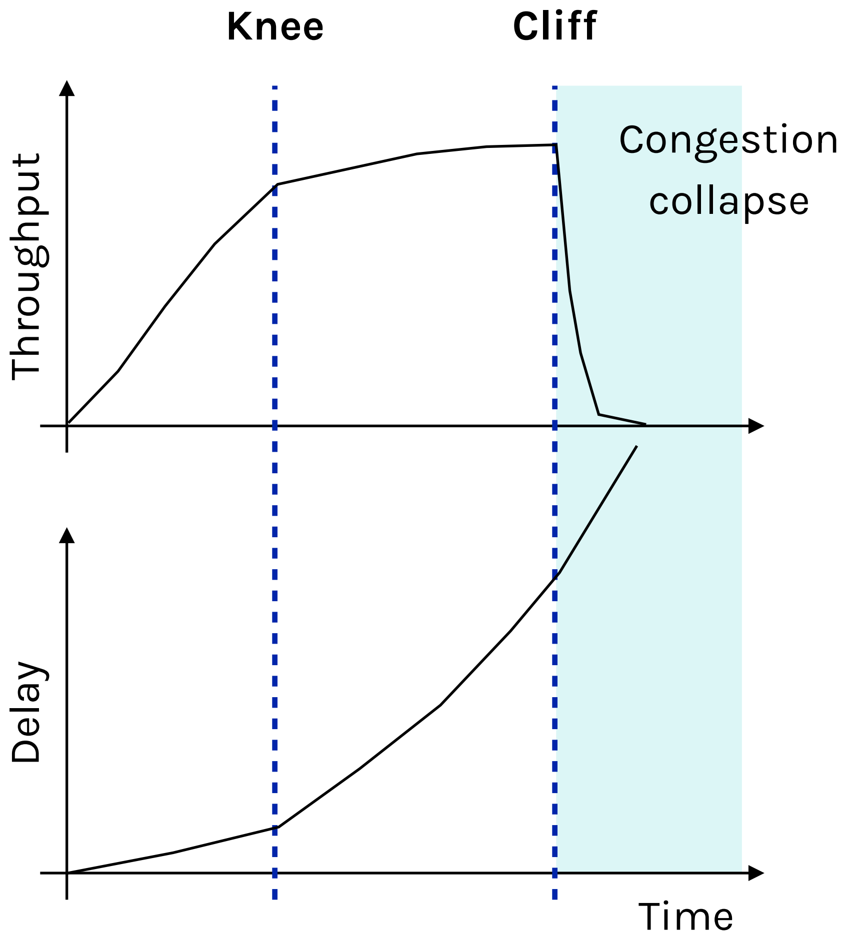


Congestion collapse

Increase in network load results in a decrease of useful work done



Congestion collapse analysis

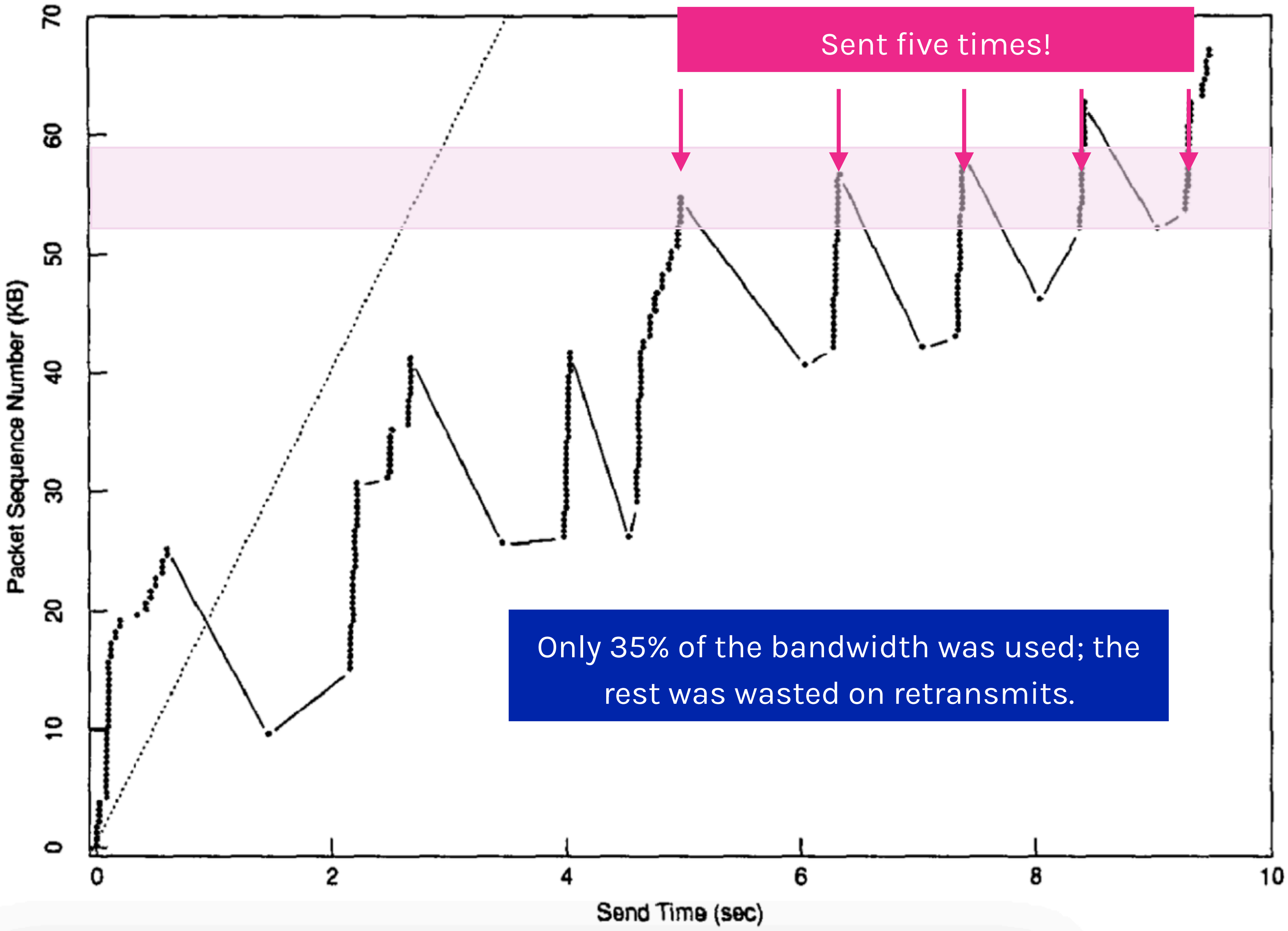


Knee point: after which

Throughput: increases slowly
Delay: increases quickly

Cliff point: after which

Throughput: decreases quickly
Delay: tends to infinity



Sent five times!
Only 35% of the bandwidth was used; the rest was wasted on retransmits.
Packet Sequence Number (KB)
0
10
20
30
40
50
60
70
Send Time (sec)
0
2
4
6
8
10

Congestion control goals

Bandwidth estimation

- How to adjust the bandwidth of a single flow to the bottleneck bandwidth?

Bandwidth adaptation

- How to adjust the bandwidth of a single flow to the variation of the bottleneck bandwidth?

Fairness

- How to share bandwidth "fairly" among flows, without overloading the network

Congestion control vs. flow control

Flow control	Congestion control
Prevents **one fast sender** from overloading a slow **receiver** (Solved using a **receiving window**)	Prevents **a set of senders** from overloading the **network** (Solved using a **congestion window**)

Windows at the sender

Receiving window $RWND$

- How many bytes can be sent without overflowing the receiver buffer?
- Based on the receiver input

Congestion window $CWND$

- How many bytes can be sent without overflowing the routers?
- Based on network conditions

Sender window: $\min\{RWND, CWND\}$

Congestion Detection and Control

Congestion detection approaches

Approach 1

- Ask the network to tell the source
- But signal itself could be lost

Approach 2

- Measure packet delay
- But signal is noisy (delay often varies considerably)

Approach 3

- Measure packet loss (fail-safe signal that TCP has to detect already)

Two signals for detecting packet loss

Loss detection can be done using ACKs or timeouts: differ in the degree of severity

Duplicate ACKs

- Mild congestion
- Packets are still making it

Timeout

- Severe congestion
- Multiple consequent losses

Reacting to congestion

What increase/decrease function should we use to adjust the $CWND$?

Bandwidth estimation
(slow-start)

Bandwidth adaptation
(additive vs. multiplicative)

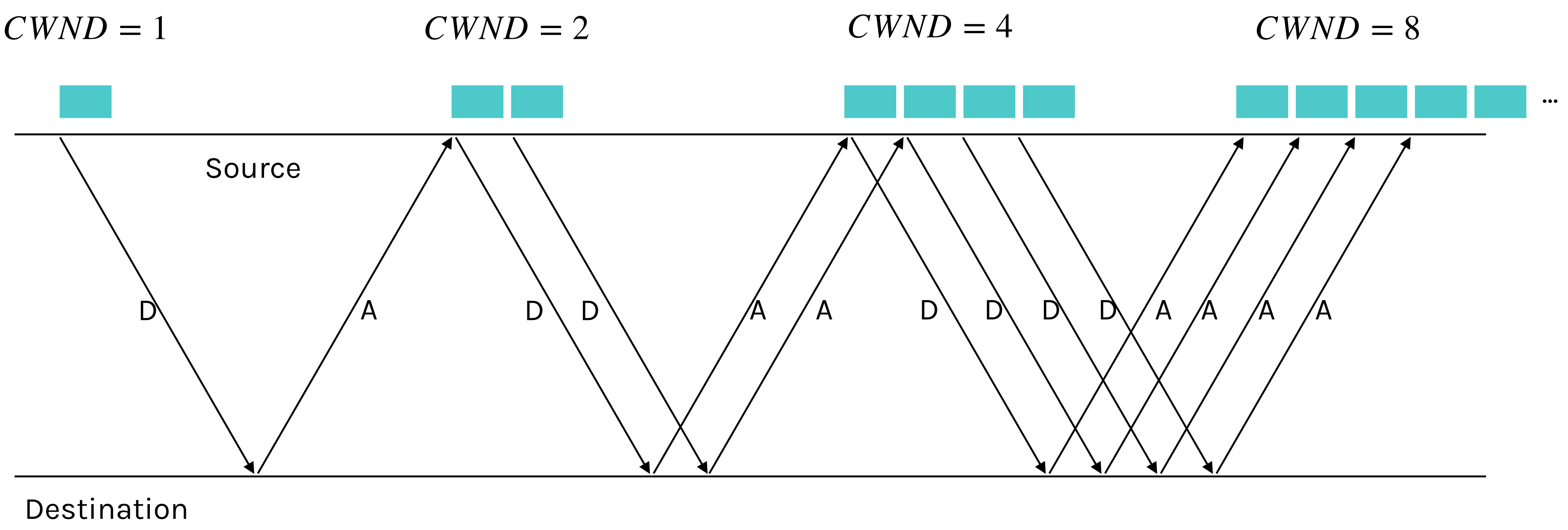
Fairness
(convergence to fair share)

Bandwidth estimation

Goal: quickly get a first-order estimate of the available bandwidth

Intuition	Start slow but rapidly increase unitl a packet drop occurs
Increase policy	$CWND = 1$ (initially) $CWND += 1$ (upon receipt of an ACK)

TCP slow-start



$CWND$ increases exponentially; slow-start is called like this only because of the small starting point!

Problem with slow-start

Problem: it can result in a full window of packet losses

Example

- Assume that $CWND$ is just enough to "fill the pipe"
- After one RTT, $CWND$ has doubled; all the excess packets are now dropped

Solution

- We need a more **gentle adjustment algorithm** once we have a rough estimate of the bandwidth

Bandwidth adaptation

Goal: track the available bandwidth and oscillate round its current value

Increase \ Decrease	**Additive**	**Multiplicative**
Additive	AIAD	AIMD
Multiplicative	MIAD	MIMD

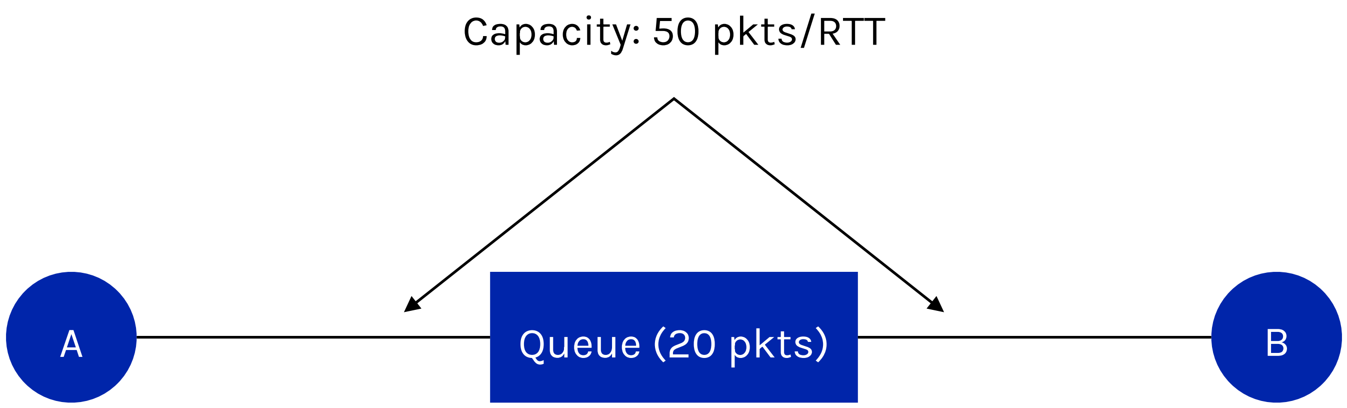
Design space for $CWND$ adjustment: 4 alternative designs

Comparison of design choices

	Increase behavior	Decrease behavior
AIAD	Gentle	Gentle
AIMD	Gentle	Aggressive
MIAD	Aggressive	Gentle
MIMD	Aggressive	Aggressive

How to select among these? Need to weight in another problem: fairness

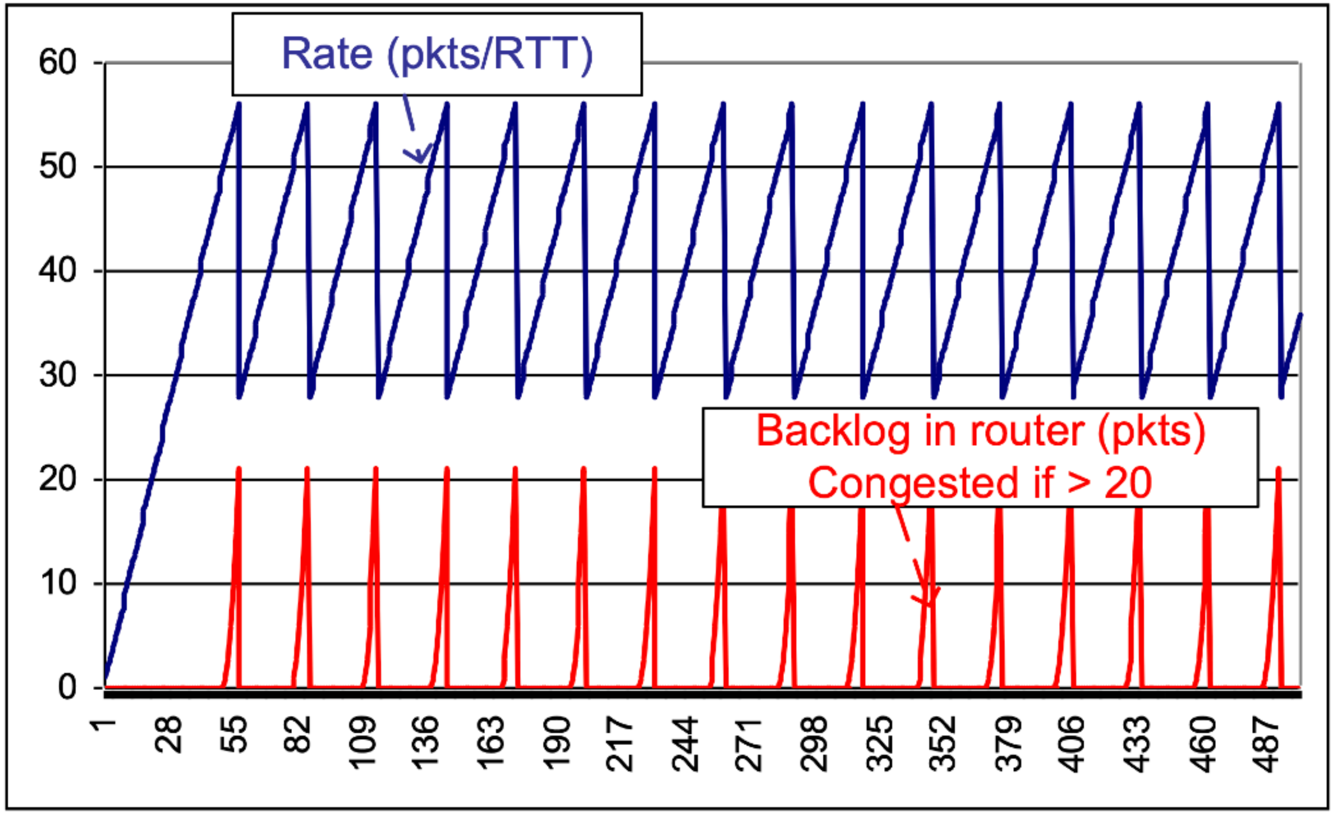
Example: one flow with AIMD



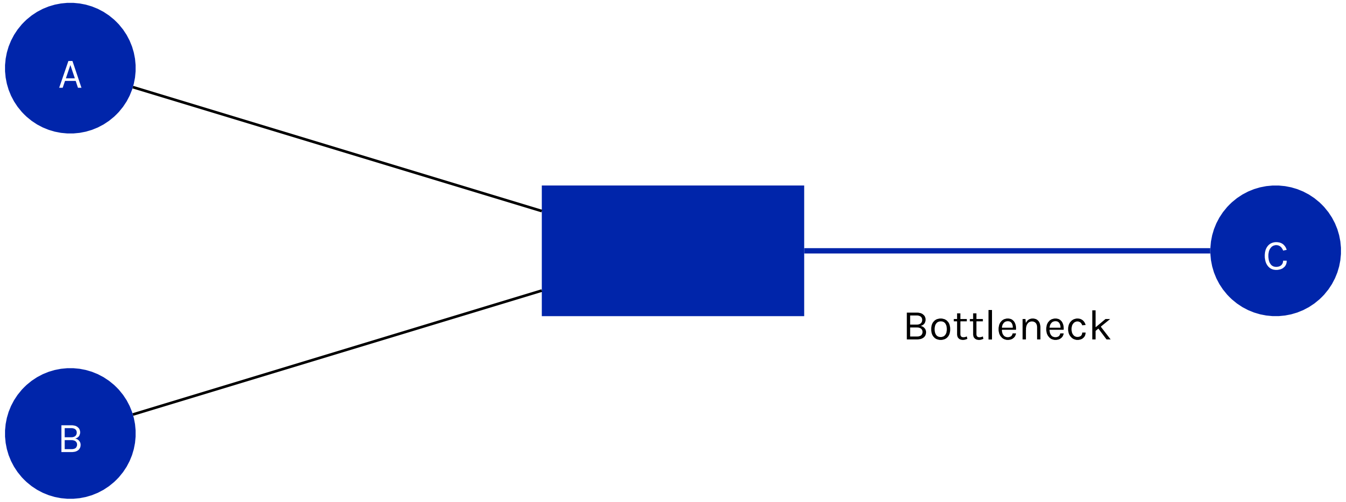
Without congestion $CWND$ increases by one packet every RTT

Upon congestion $CWND$ decreases by a factor of 2

Example: one flow with AIMD

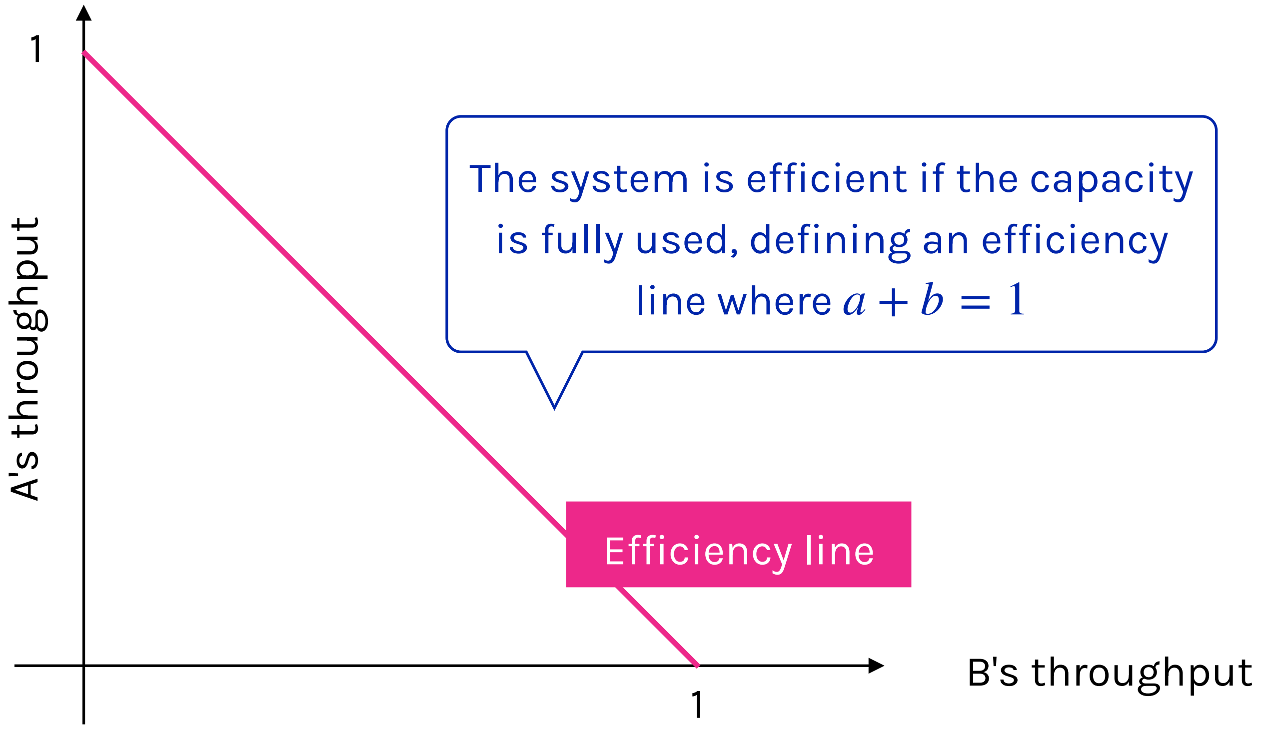


Example: two flows sharing a bottleneck



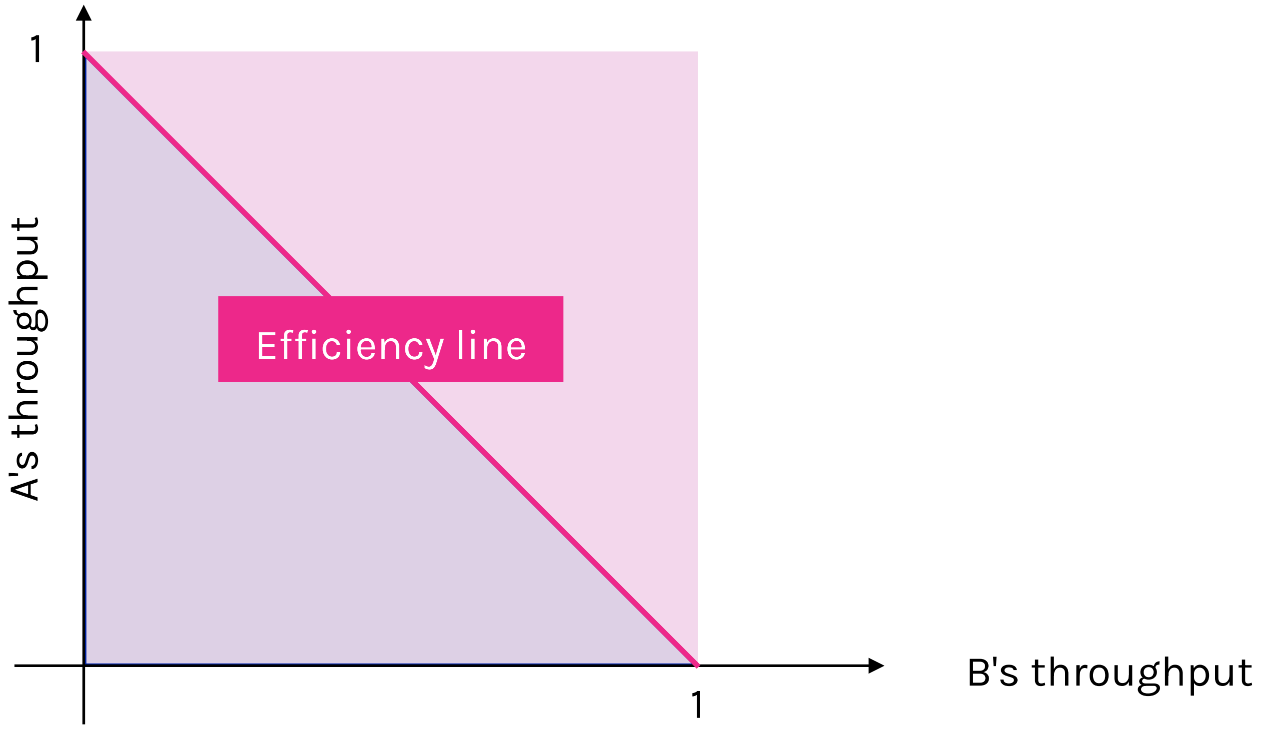
System behavior

System behavior: efficiency line

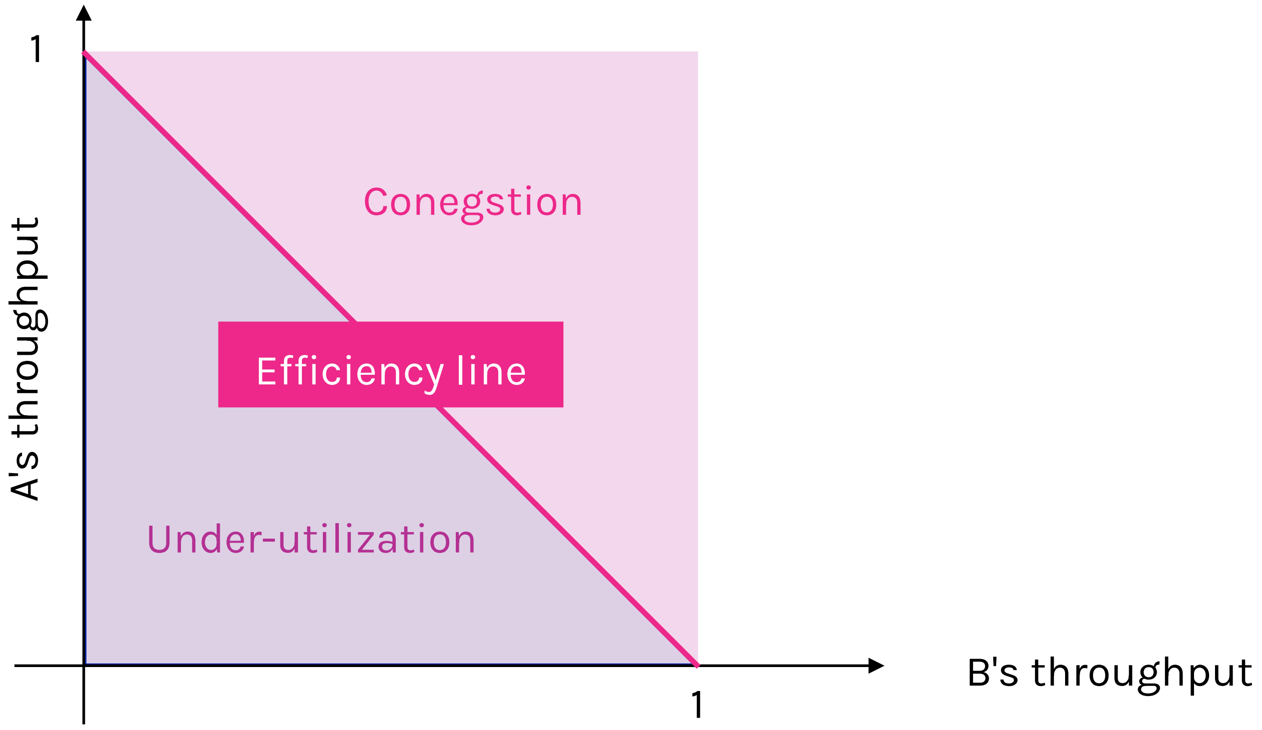


Goal of congestion control: bring the system as close as possible to this line and stay there

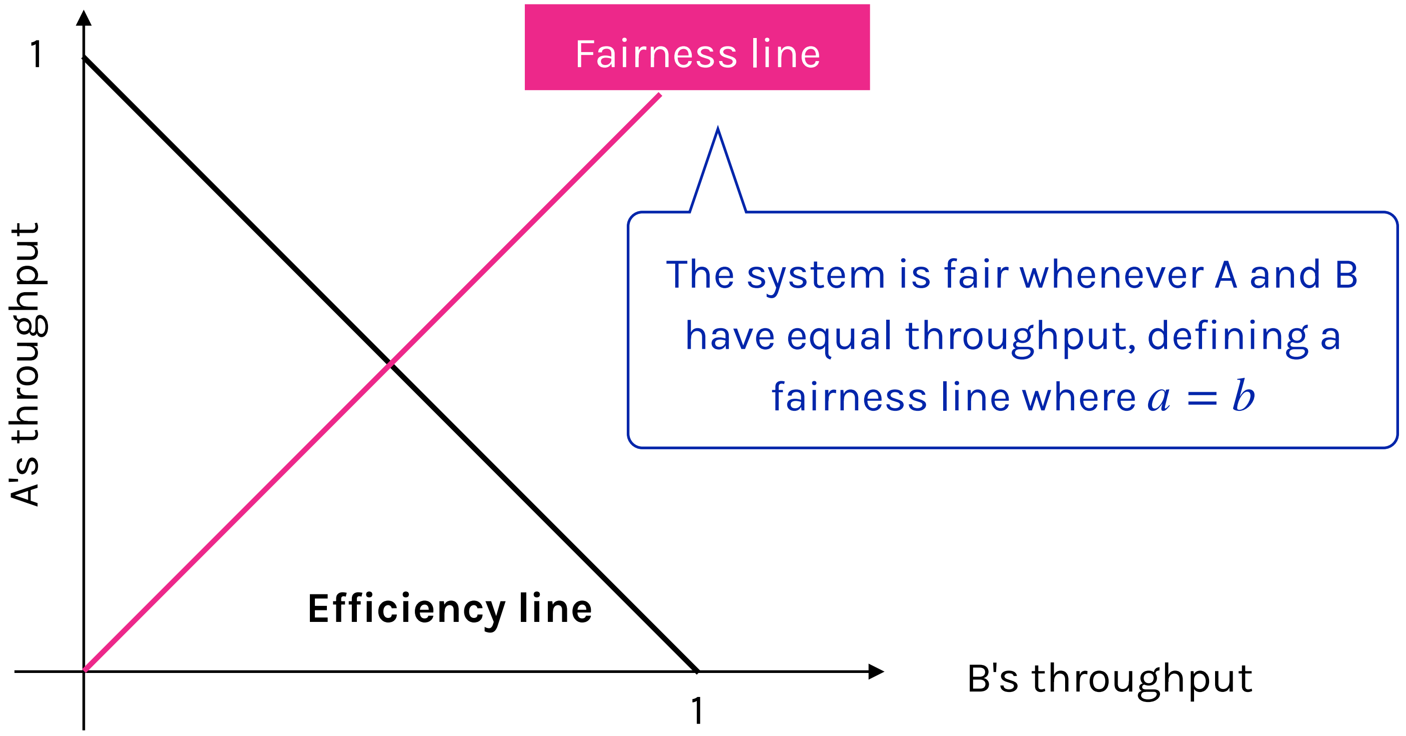
System behavior: efficiency line



System behavior: efficiency line

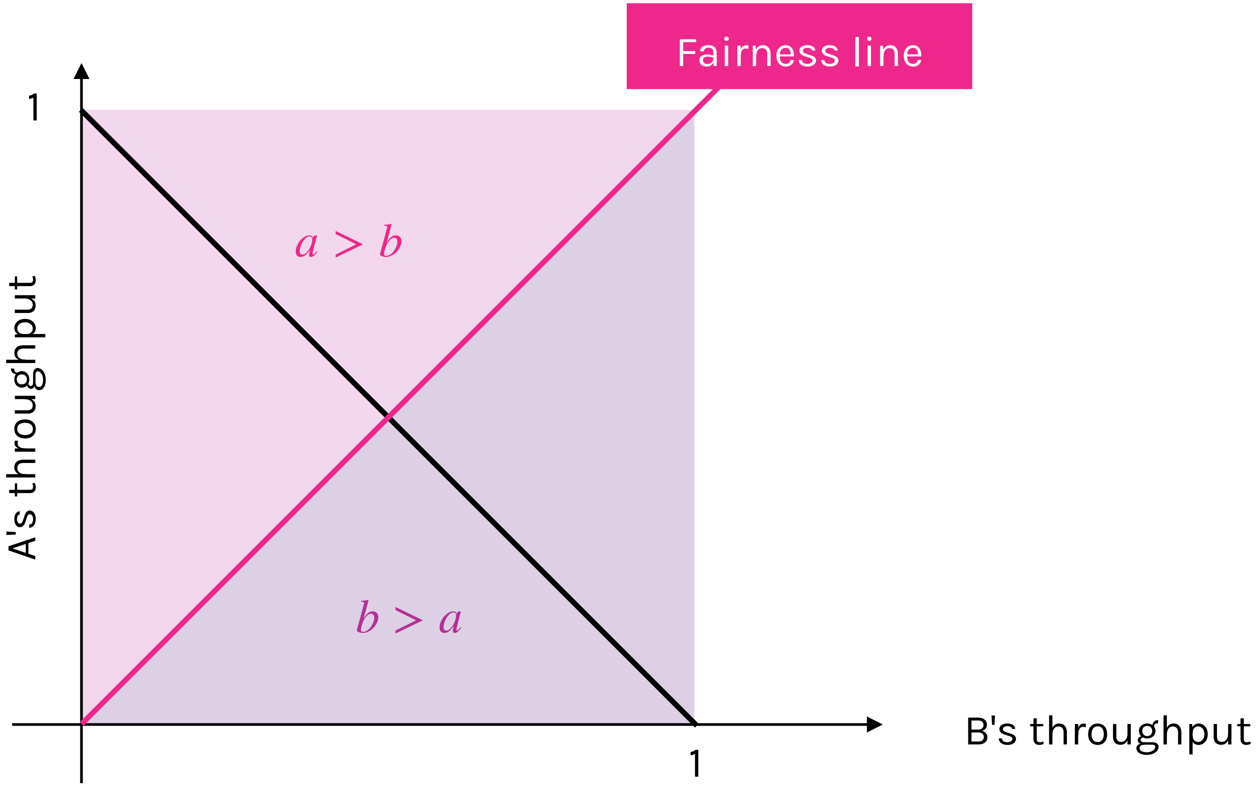


System behavior: fairness line

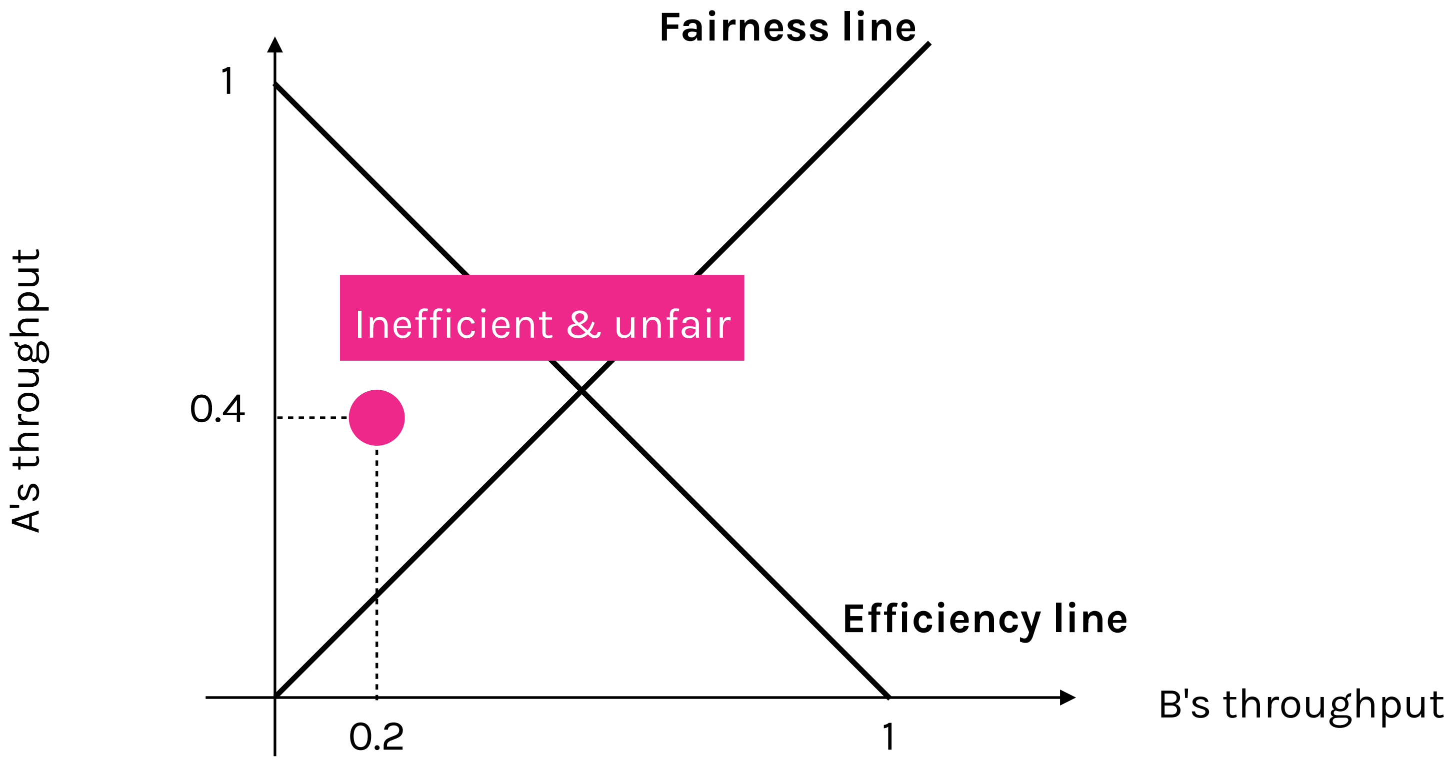


Goal of congestion control: bring the system as close as possible to this line and stay there

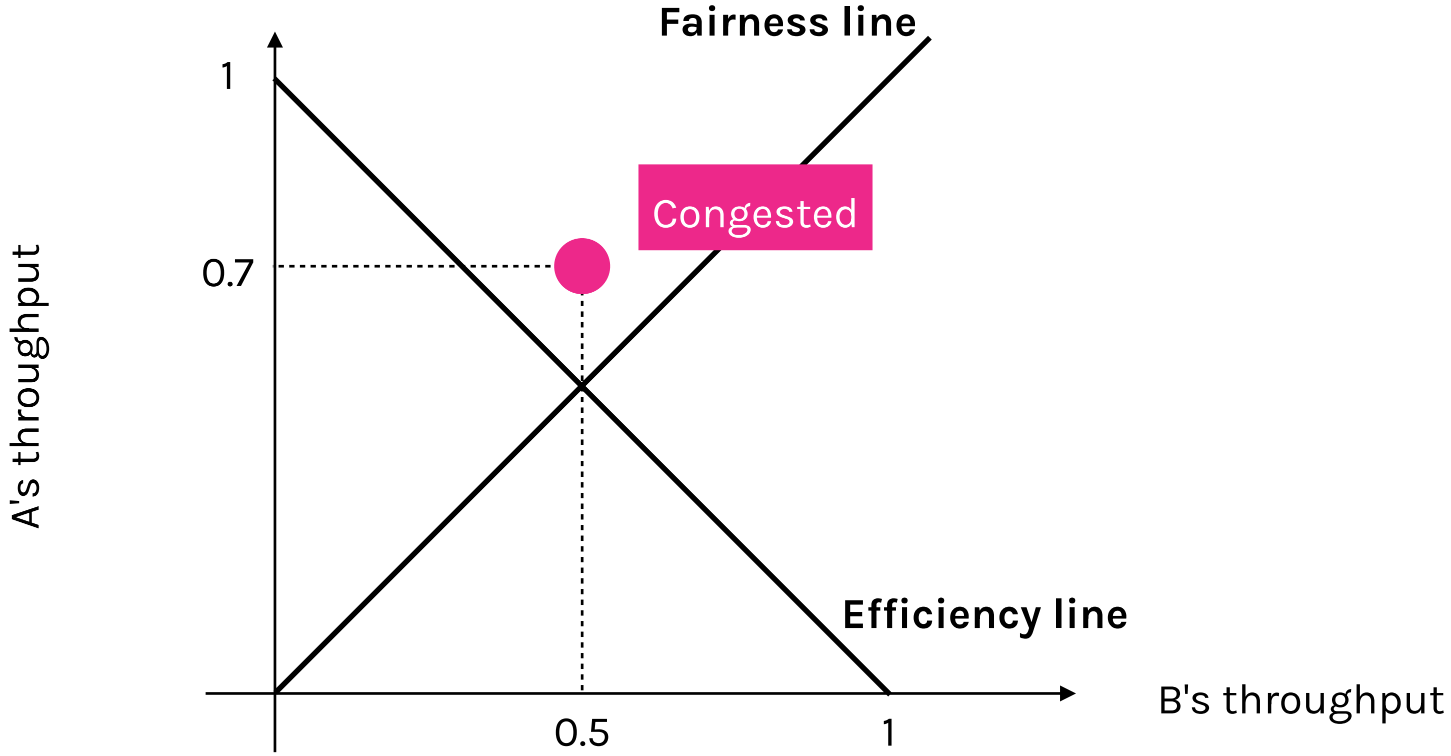
System behavior: fairness line



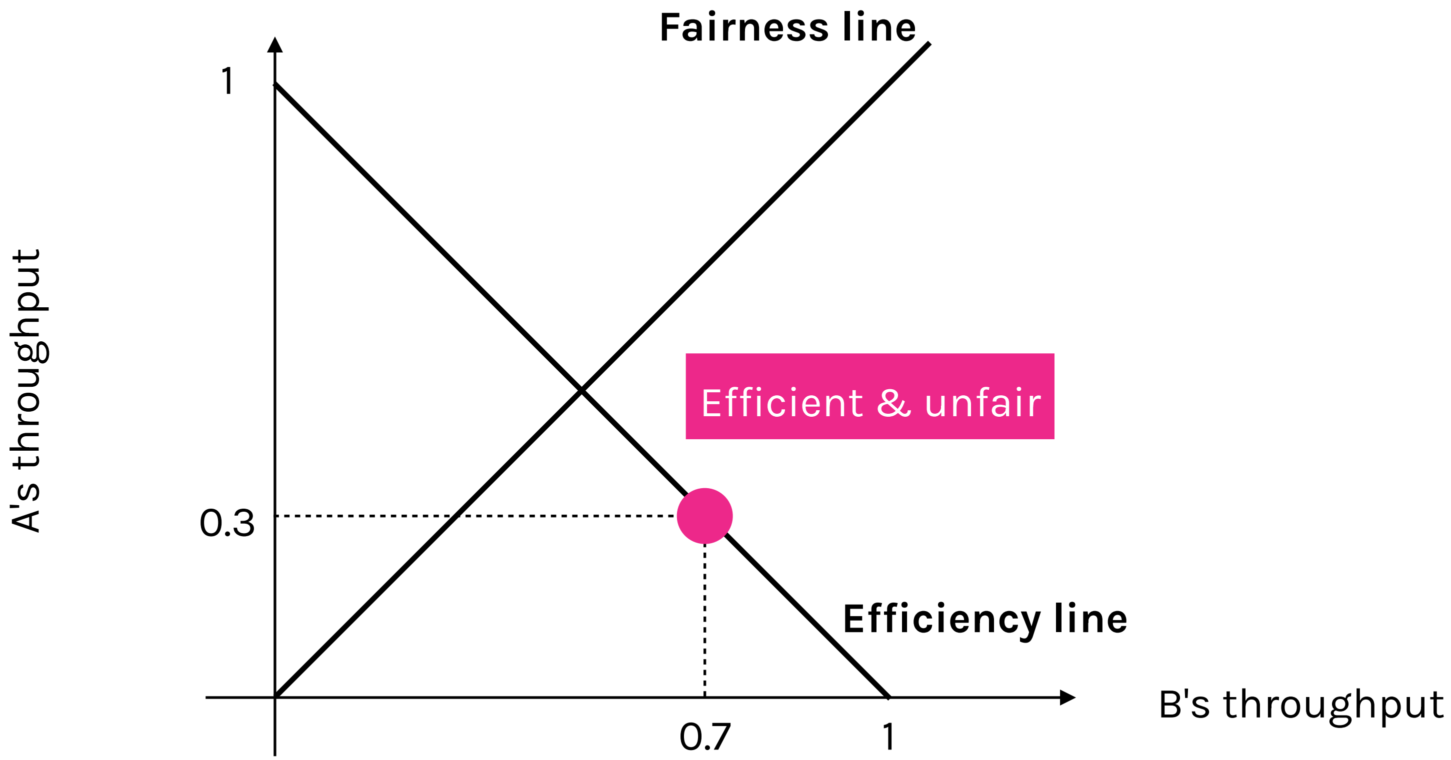
System behavior: possible states



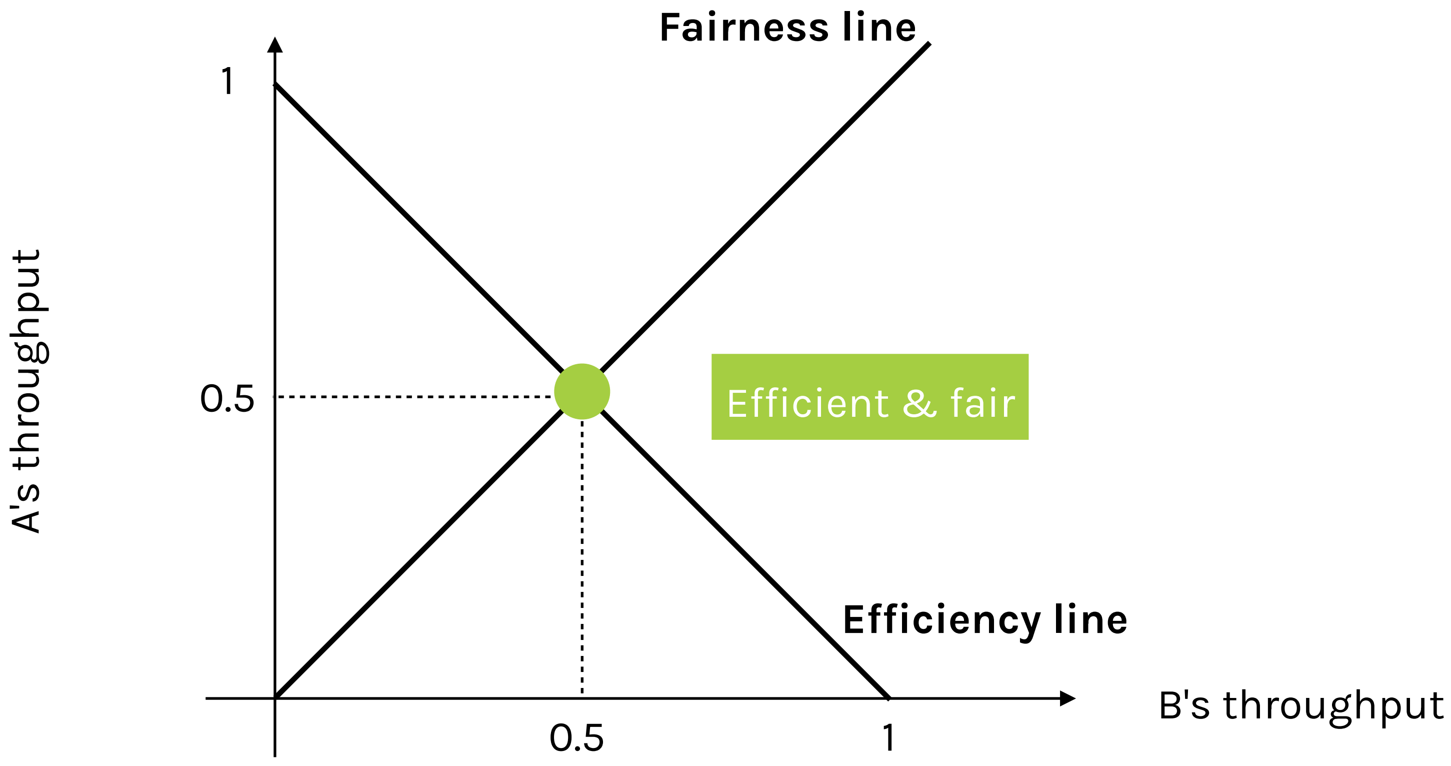
System behavior: possible states



System behavior: possible states



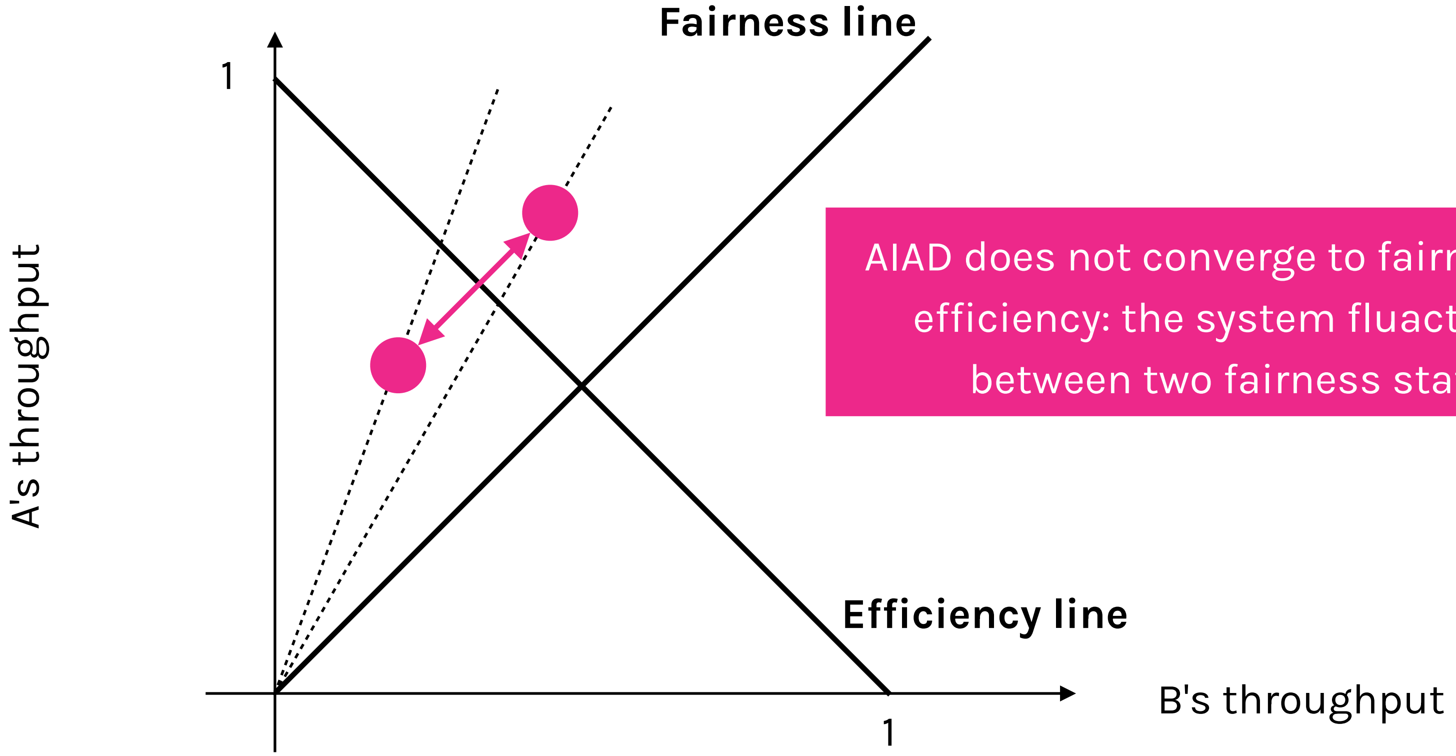
System behavior: possible states



Comparison of design choices

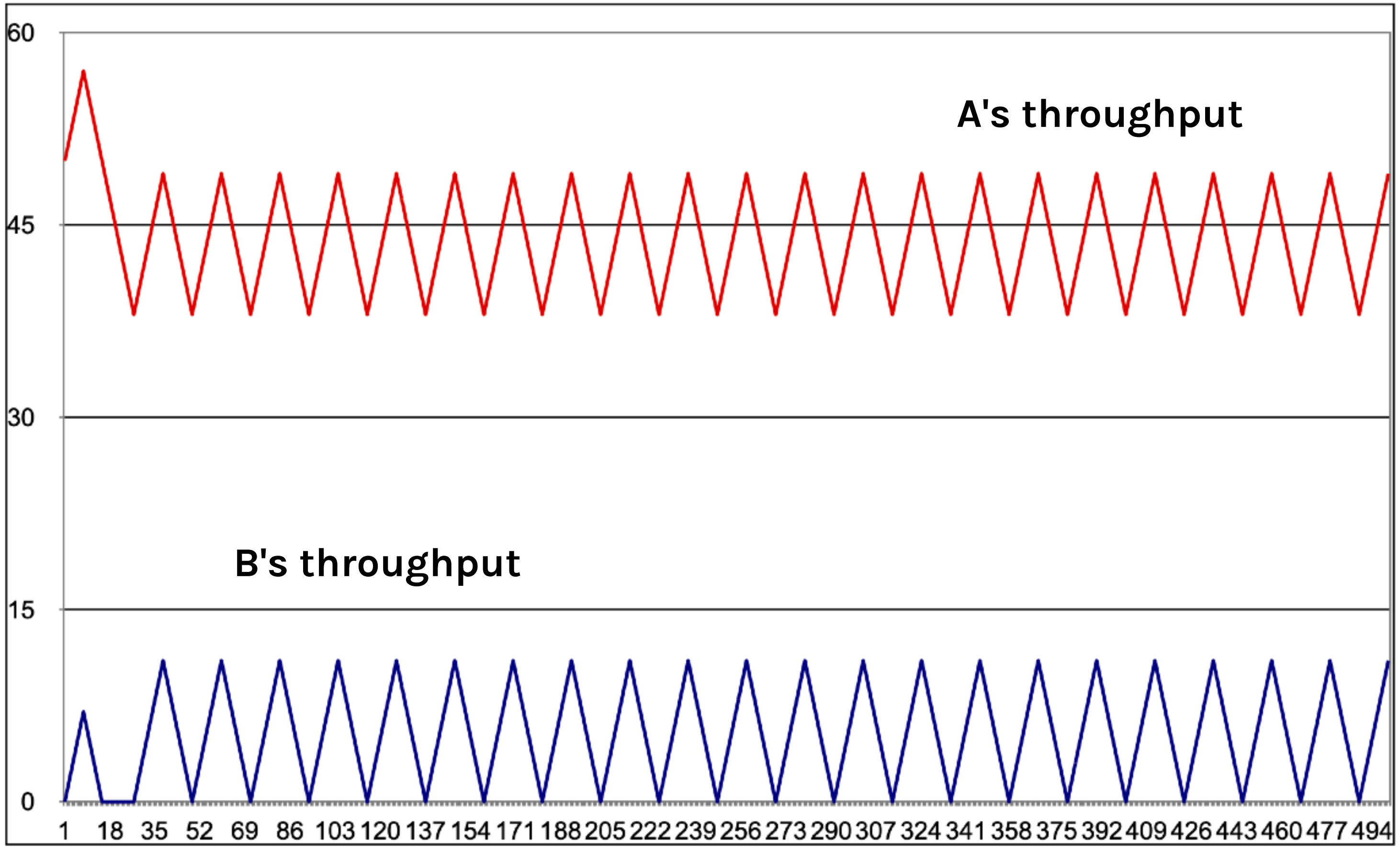
	Increase behavior	Decrease behavior
AIAD	Gentle	Gentle
AIMD	Gentle	Aggressive
MIAD	Aggressive	Gentle
MIMD	Aggressive	Aggressive

AIAD fairness



AIAD does not converge to fairness, nor efficiency: the system fluactuates between two fairness states

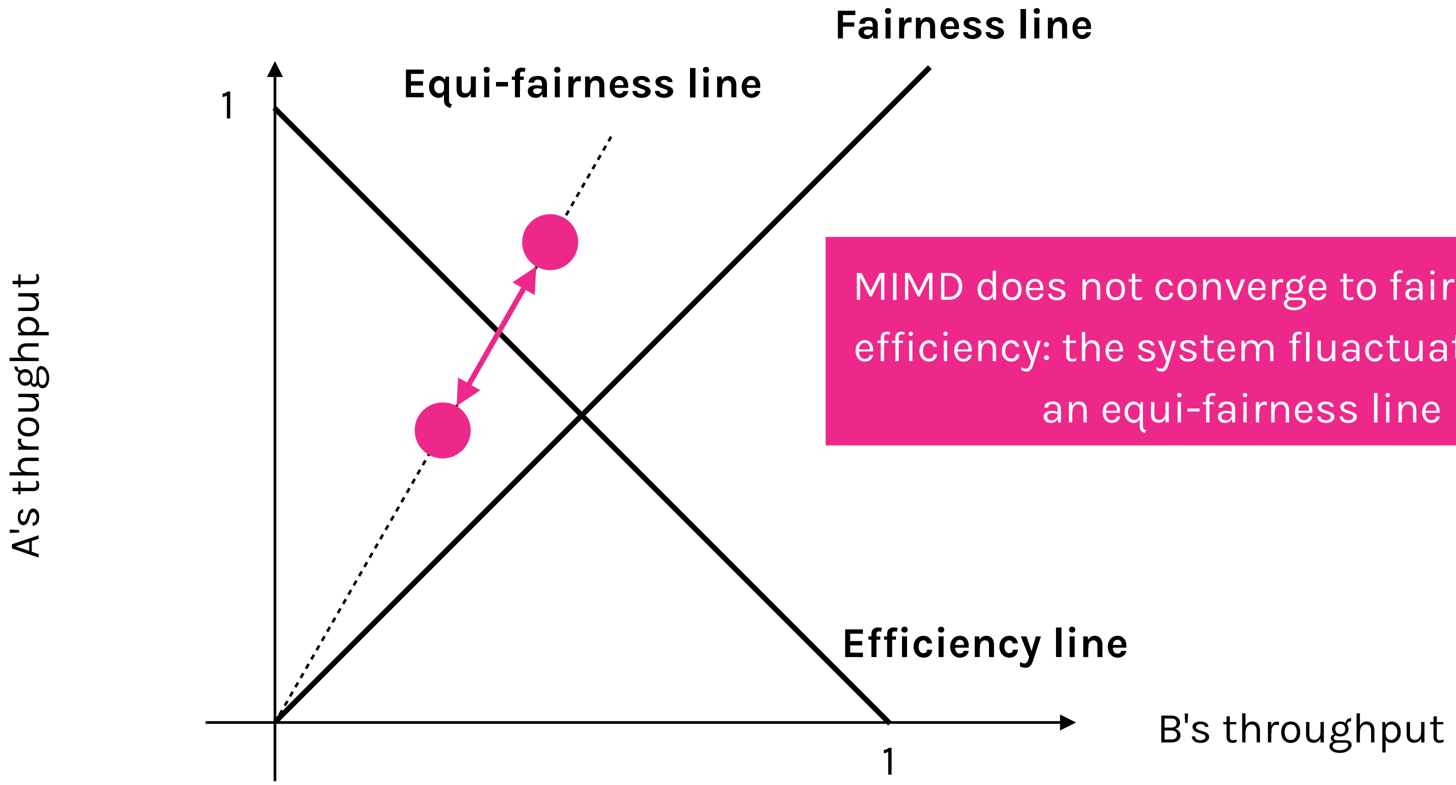
AIAD fairness



Comparison of design choices

	Increase behavior	Decrease behavior
AIAD	Gentle	Gentle
AIMD	Gentle	Aggressive
MIAD	Aggressive	Gentle
MIMD	Aggressive	Aggressive

MIMD fairness

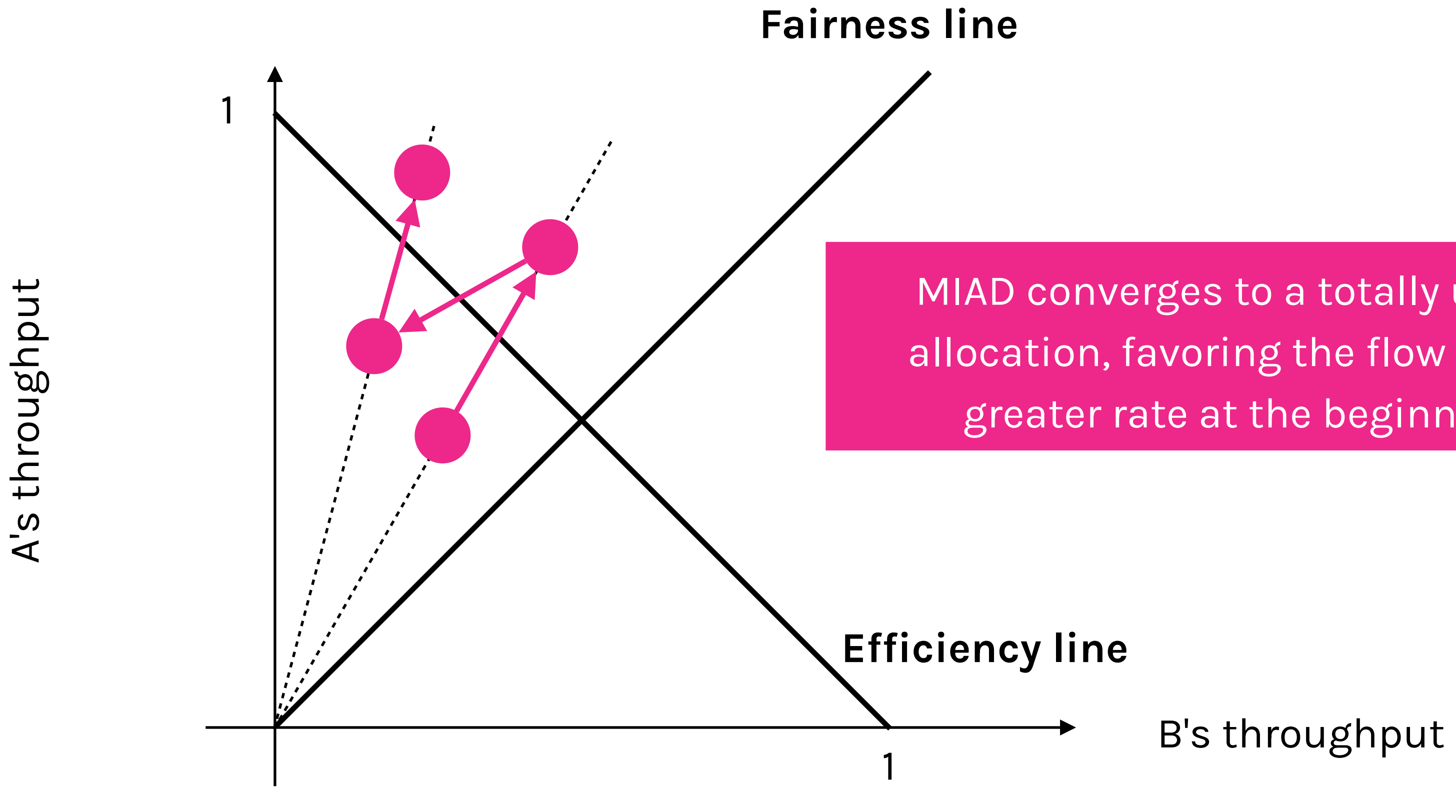


MIMD does not converge to fairness, nor efficiency: the system fluactuates along an equi-fairness line

Comparison of design choices

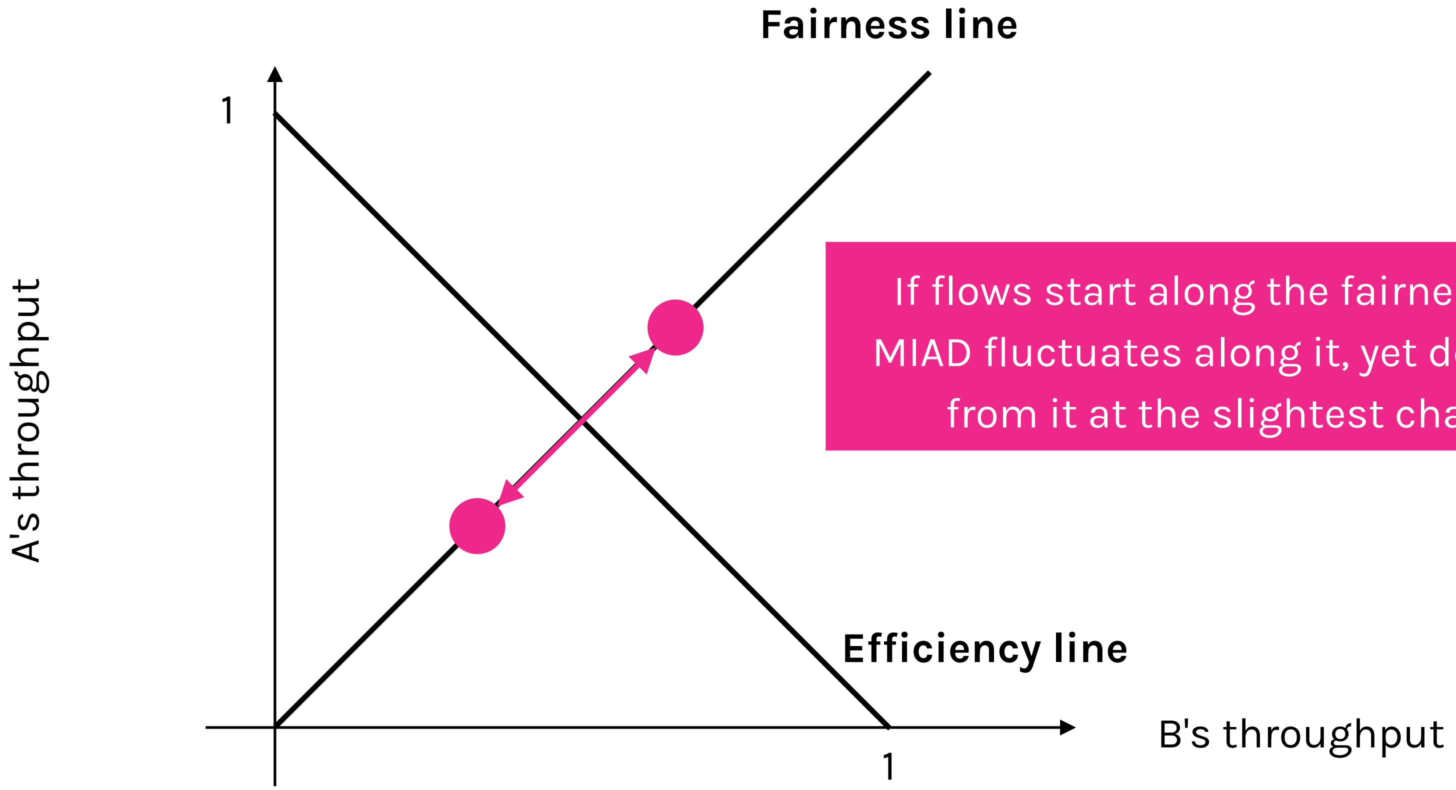
	Increase behavior	Decrease behavior
AIAD	Gentle	Gentle
AIMD	Gentle	Aggressive
MIAD	Aggressive	Gentle
MIMD	Aggressive	Aggressive

MIAD fairness



MIAD converges to a totally unfair allocation, favoring the flow with a greater rate at the beginning

MIAD fairness

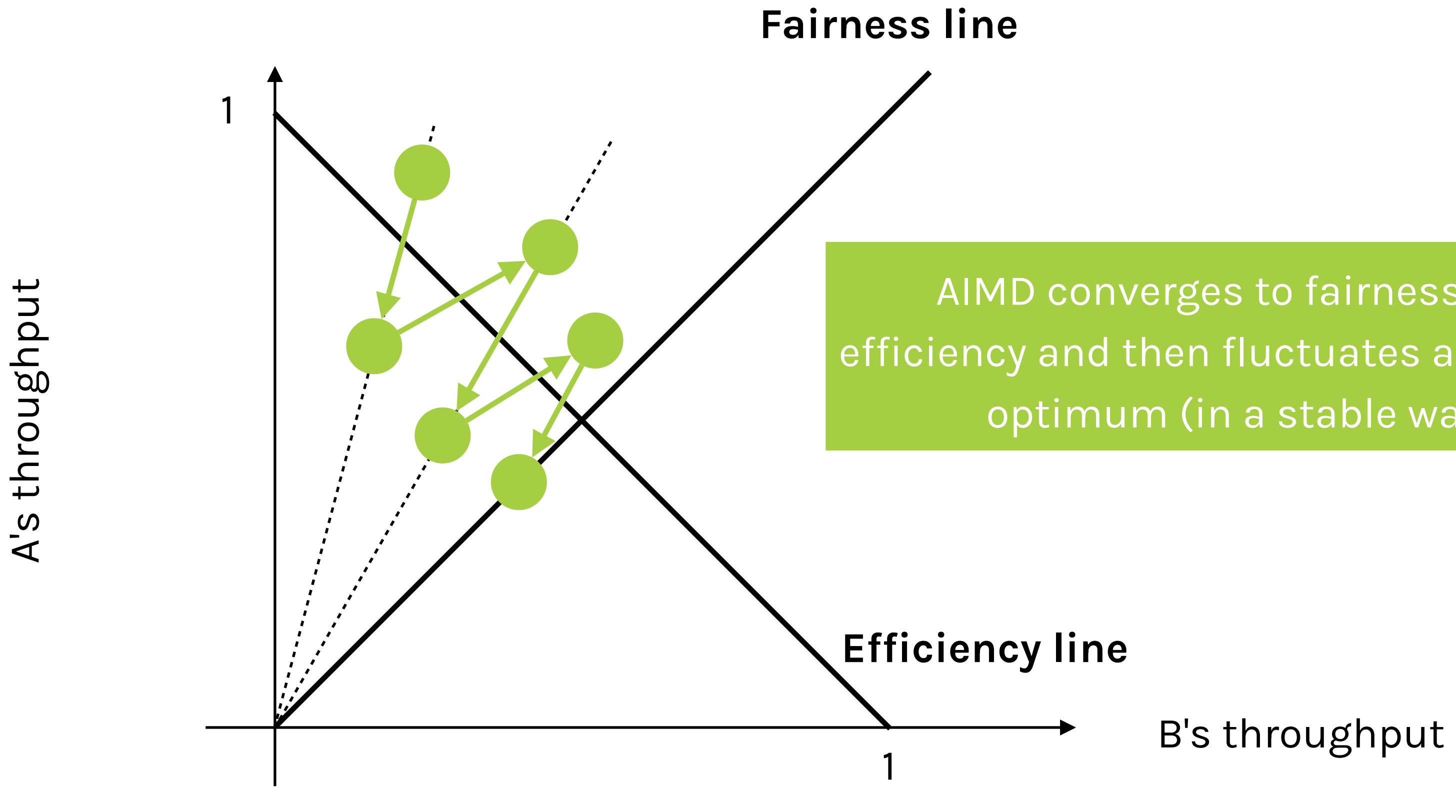


If flows start along the fairness line, MIAD fluctuates along it, yet deviating from it at the slightest change

Comparison of design choices

	Increase behavior	Decrease behavior
AIAD	Gentle	Gentle
AIMD	Gentle	Aggressive
MIAD	Aggressive	Gentle
MIMD	Aggressive	Aggressive

AIMD fairness



AIMD converges to fairness and efficiency and then fluctuates around the optimum (in a stable way)

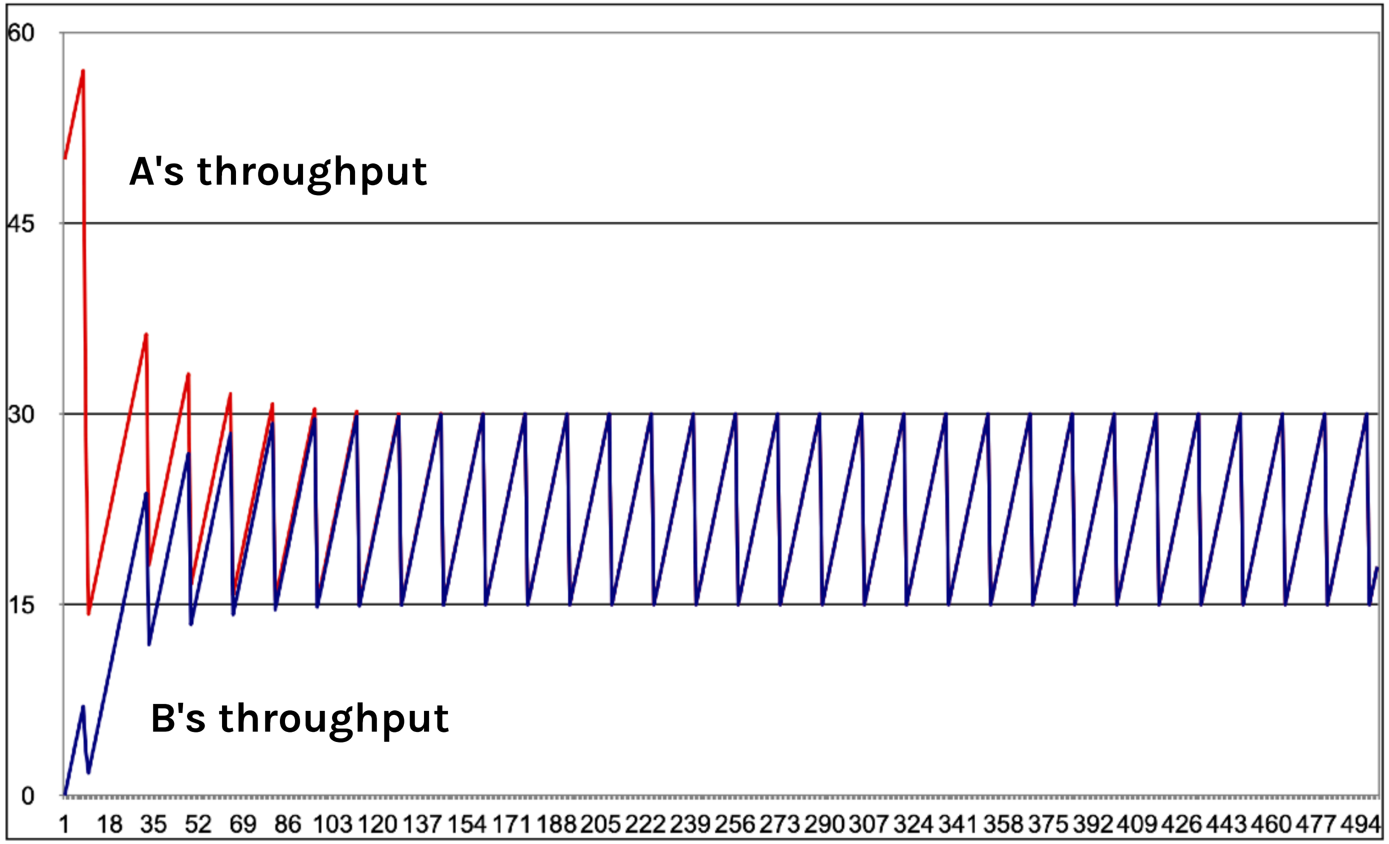
AIMD fairness

Intuition

During increase, both flows gain bandwidth at the same rate

During decrease, the flow with a higher rate releases more

AIMD fairness



TCP implements AIMD

Implementation

After each ACK: increment $CWND$ by $1/CWND$
(linear increase of max. 1 per RTT)

Question

When does a sender leave slow-start and start AIMD?

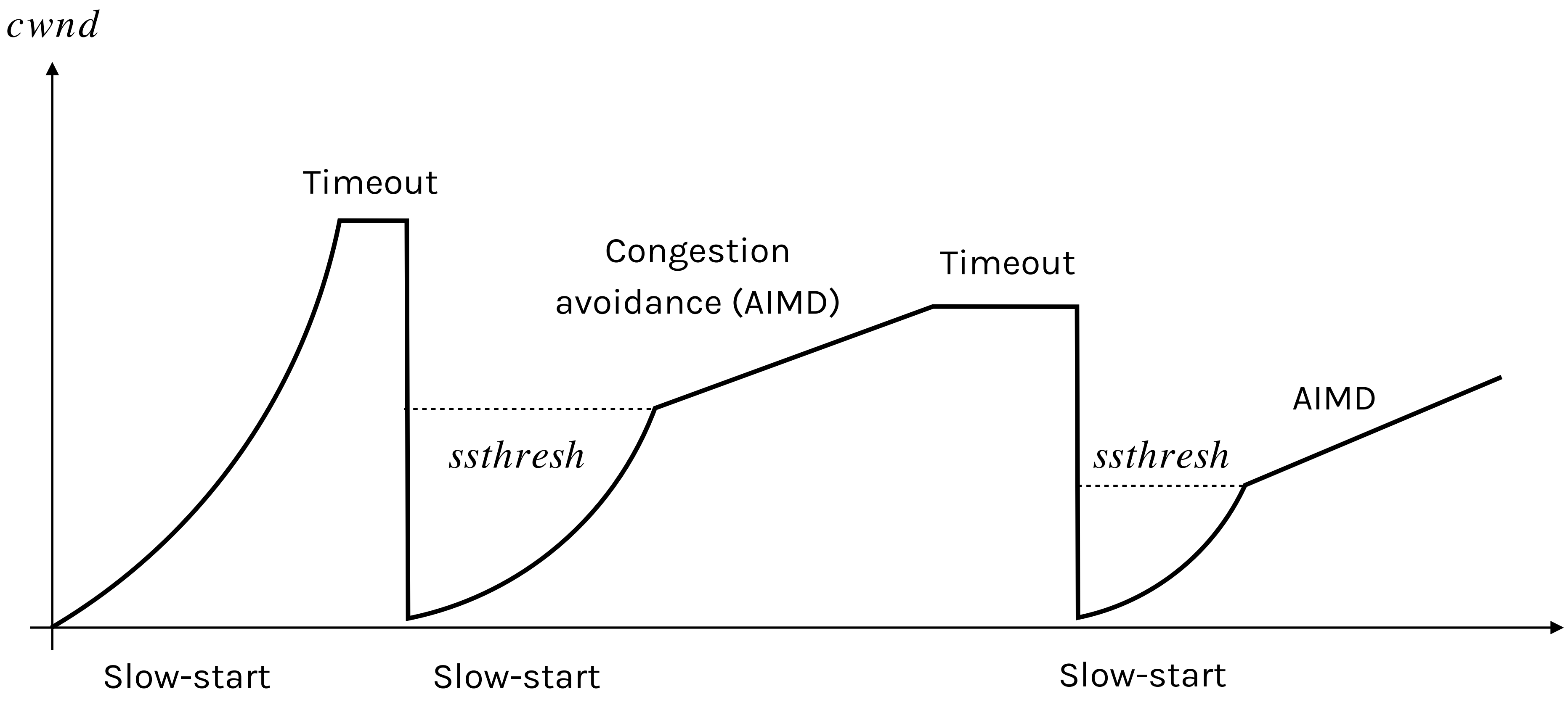
Introduce a new slow-start threshold ($ssthresh$), adapt it in function of congestion, e.g.,

$ssthresh = CWND/2$ (on timeout)

TCP congestion control

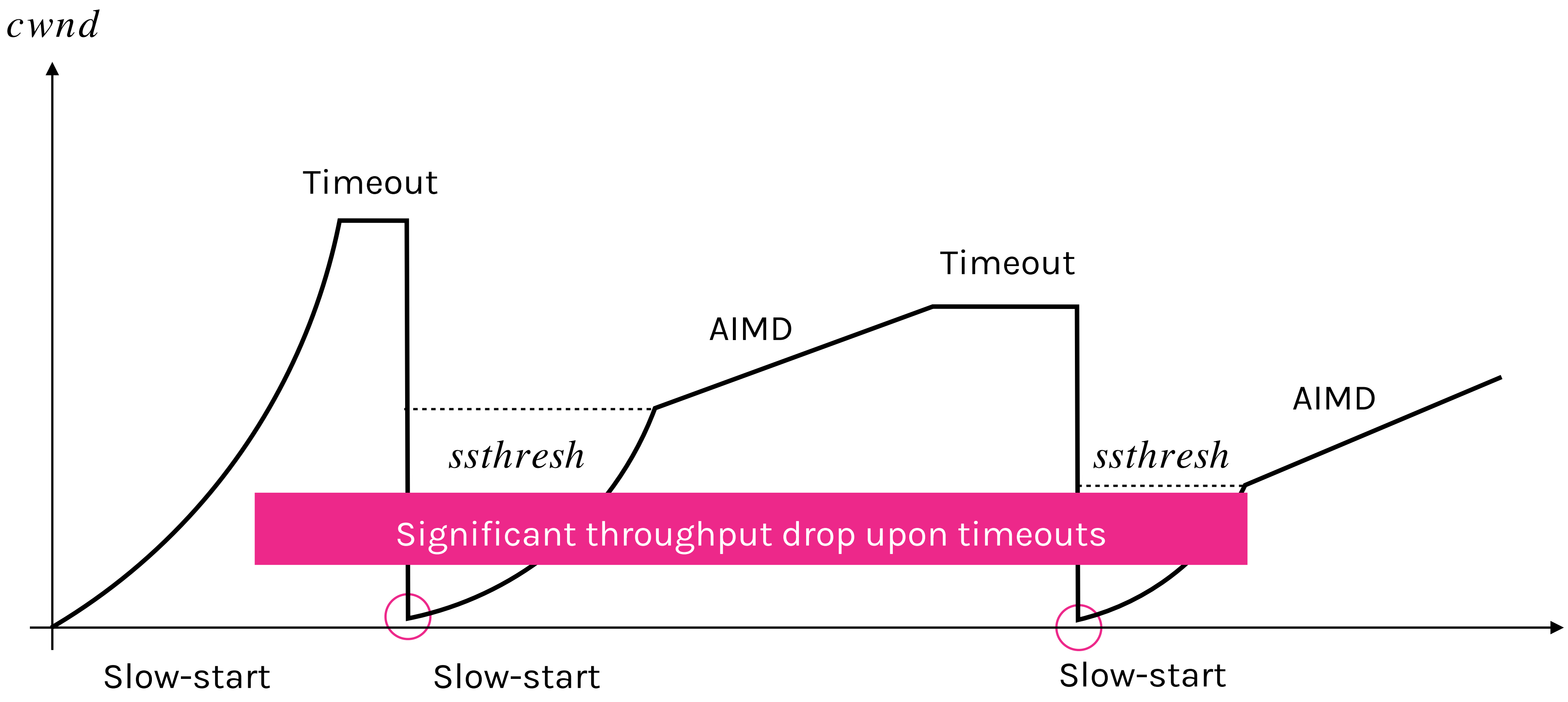
```
Initially:
    cwnd = 1
    ssthresh = infinite
New ACK received:
    if (cwnd < ssthresh):
        /* Slow Start */
        cwnd = cwnd + 1
    else:
        /* Congestion Avoidance */
        cwnd = cwnd + 1/cwnd
Timeout:
    /* Multiplicative decrease */
    ssthresh = cwnd/2
    cwnd = 1
```

TCP Tahoe



cwnd
Timeout
Congestion avoidance (AIMD)
Timeout
ssthresh
ssthresh
AIMD
Slow-start
Slow-start
Slow-start

TCP Tahoe



cwnd
Timeout
Timeout
AIMD
AIMD
ssthresh
ssthresh
Significant throughput drop upon timeouts
Slow-start
Slow-start
Slow-start

Recall: two signals for detecting packet loss

Loss detection can be done using ACKs or timeouts: differ in the degree of severity

Duplicate ACKs

- Mild congestion
- Packets are still making it

Timeout

- Severe congestion
- Multiple consequent losses

TCP fast retransmit and fast recovery

Fast retransmit

TCP automatically resends a segment after receiving 3 duplicate ACKs for it

Fast recovery

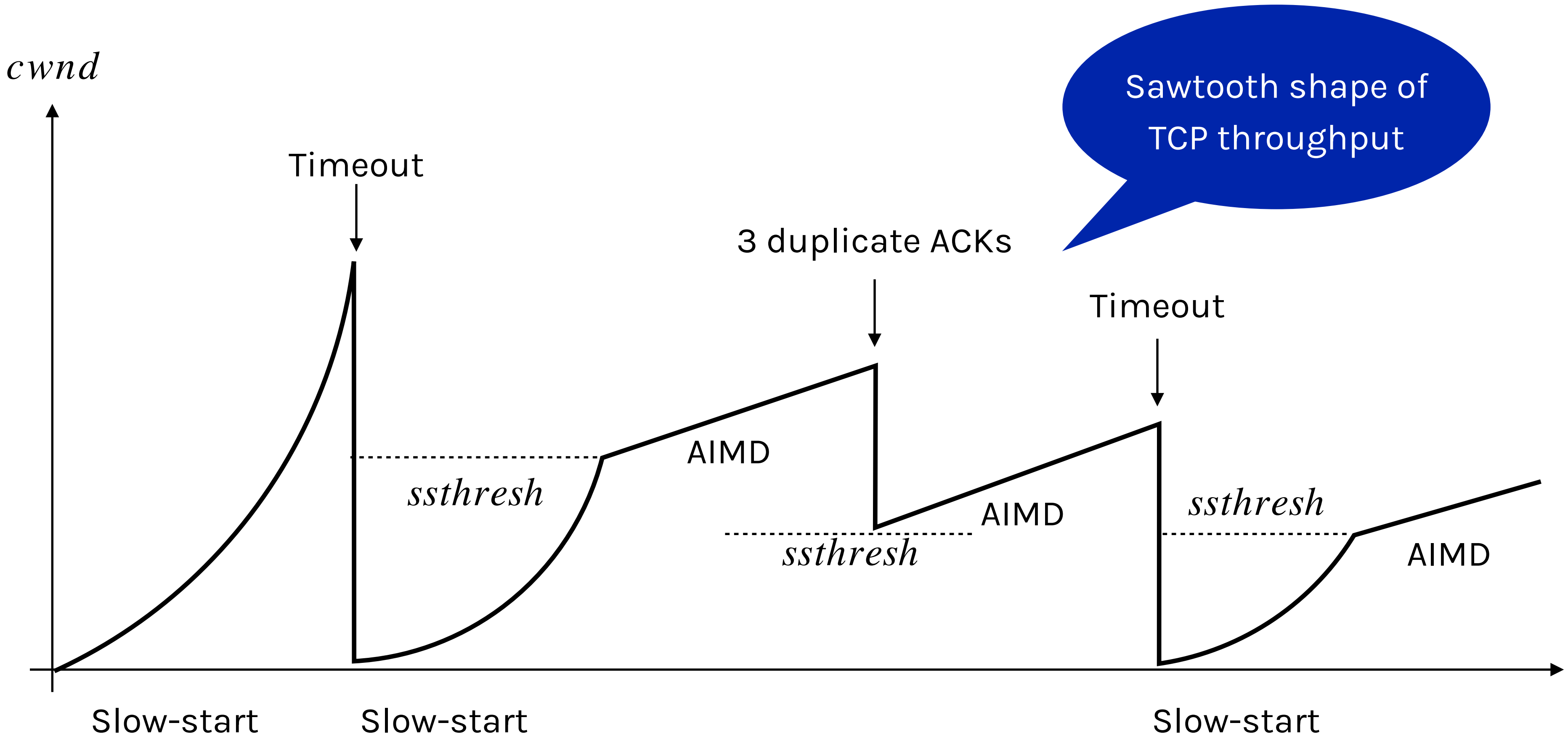
After a fast retransmit, TCP switches back to AIMD, without going all way back to 1

TCP congestion control (with fast recovery)

```
Initially:
    cwnd = 1
    ssthresh = infinite
New ACK received:
    if (cwnd < ssthresh):
        /* Slow Start */
        cwnd = cwnd + 1
    else:
        /* Congestion Avoidance */
        cwnd = cwnd + 1/cwnd
    dup_ack = 0
Timeout:
    /* Multiplicative decrease */
    ssthresh = cwnd/2
    cwnd = 1
```

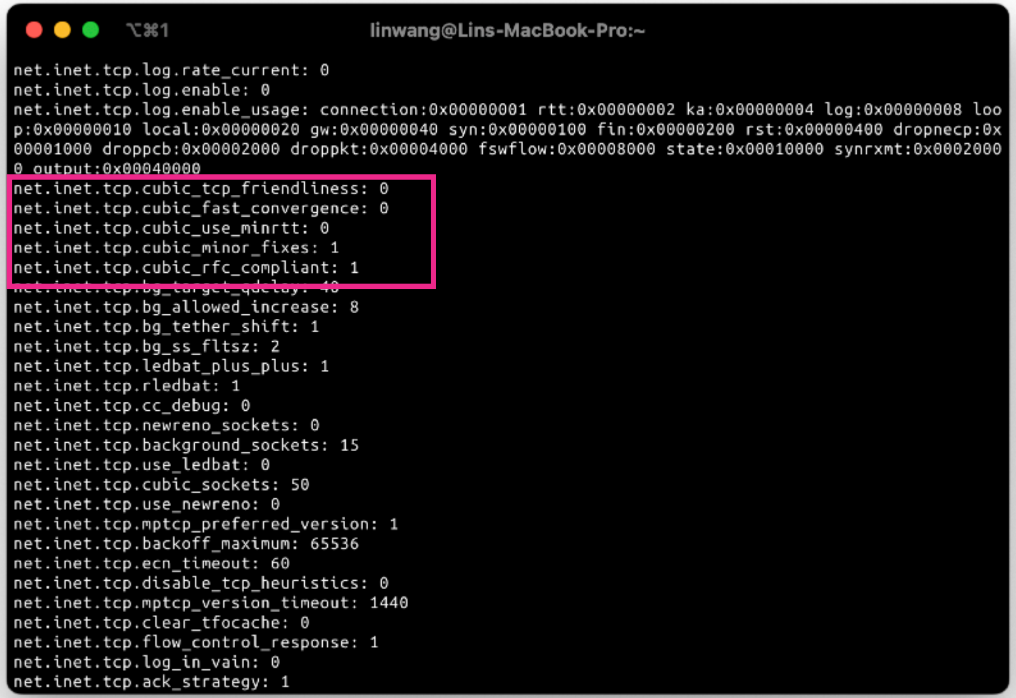
```
Duplicate ACKs received:
    dup_ack ++
    if (dup_ack >= 3):
        /* Fast recovery */
        ssthresh = cwnd/2
        cwnd = ssthresh
```

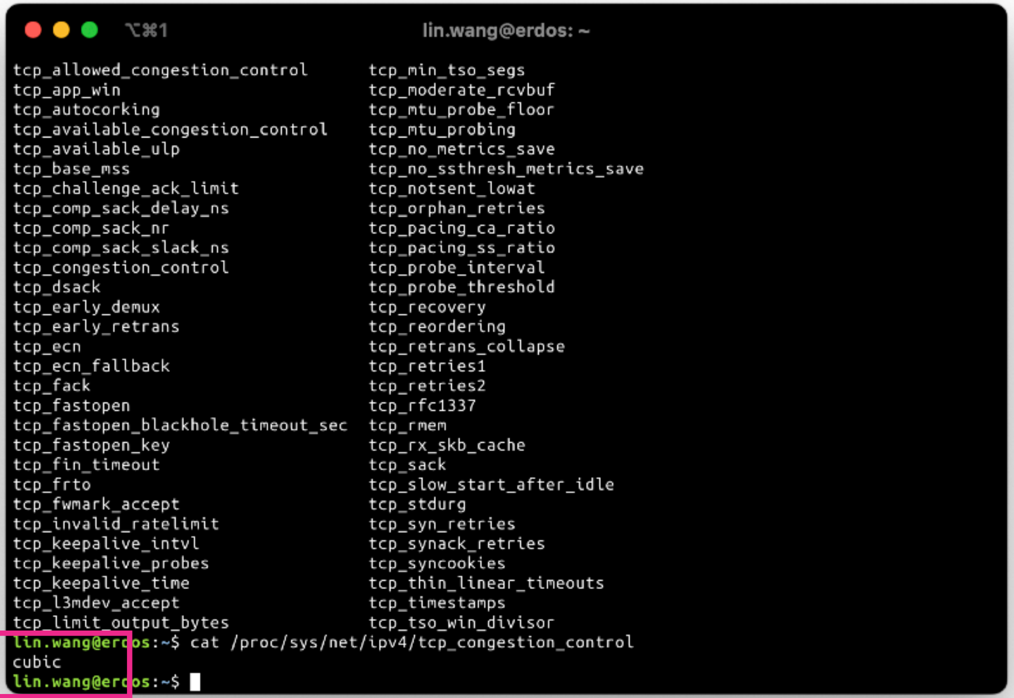
TCP Reno



cwnd
Timeout
3 duplicate ACKs
Timeout
Sawtooth shape of TCP throughput
ssthresh
AIMD
ssthresh
AIMD
ssthresh
AIMD
Slow-start
Slow-start
Slow-start

TCP CUBIC widely used

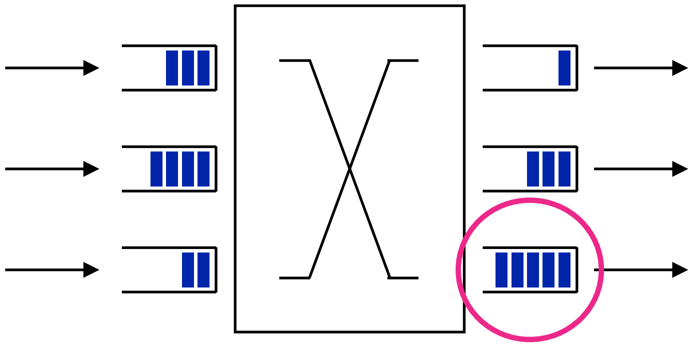




Explicit Congestion Notification (ECN)

Why congestion happens?

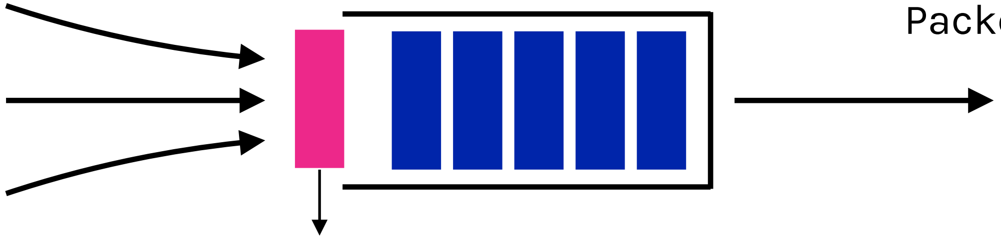
Router/switch architecture



Can we avoid the buffer to be full completely?

Drop-tail queueing

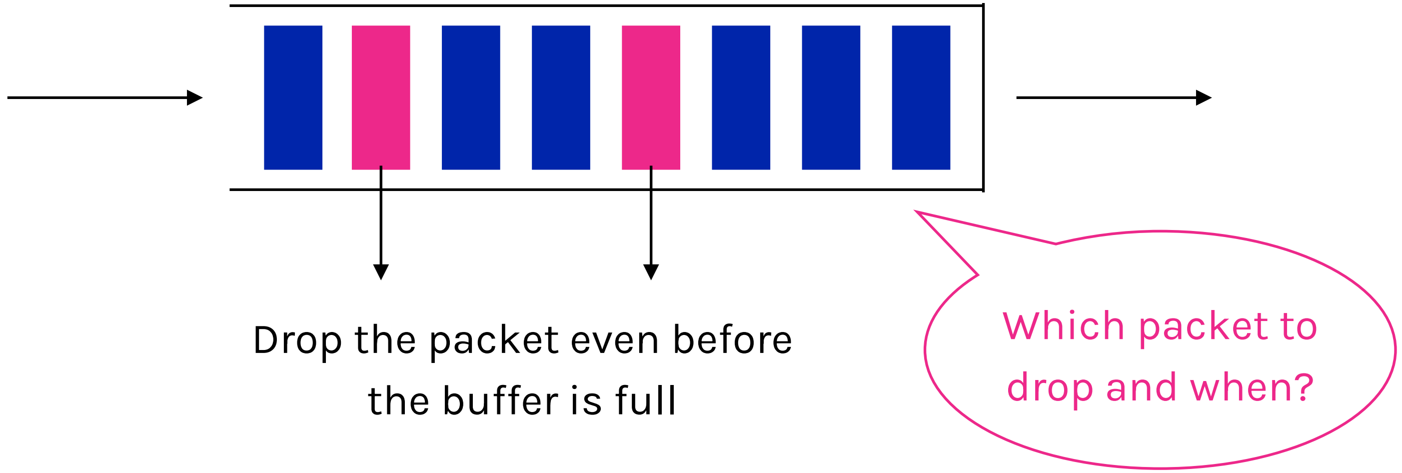
Packet arrival



Packet departure

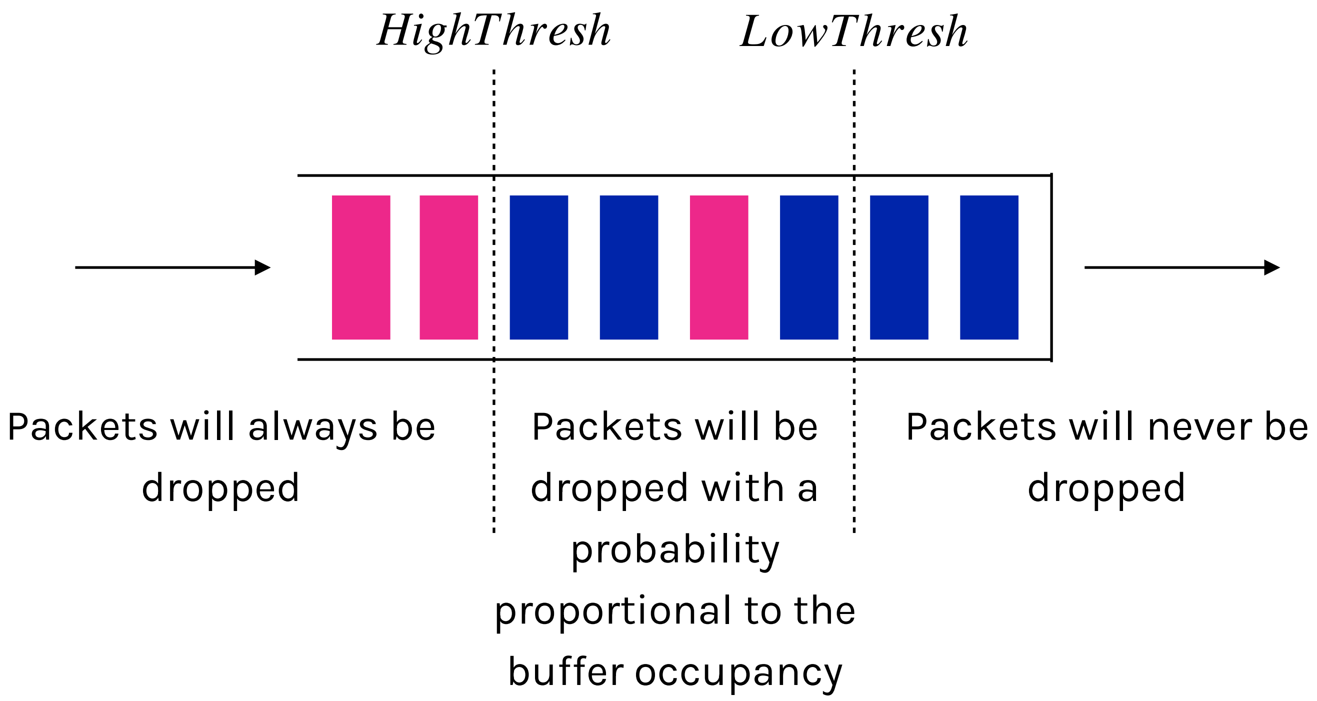
Drop the packet when the buffer is full

Active queue management (AQM)



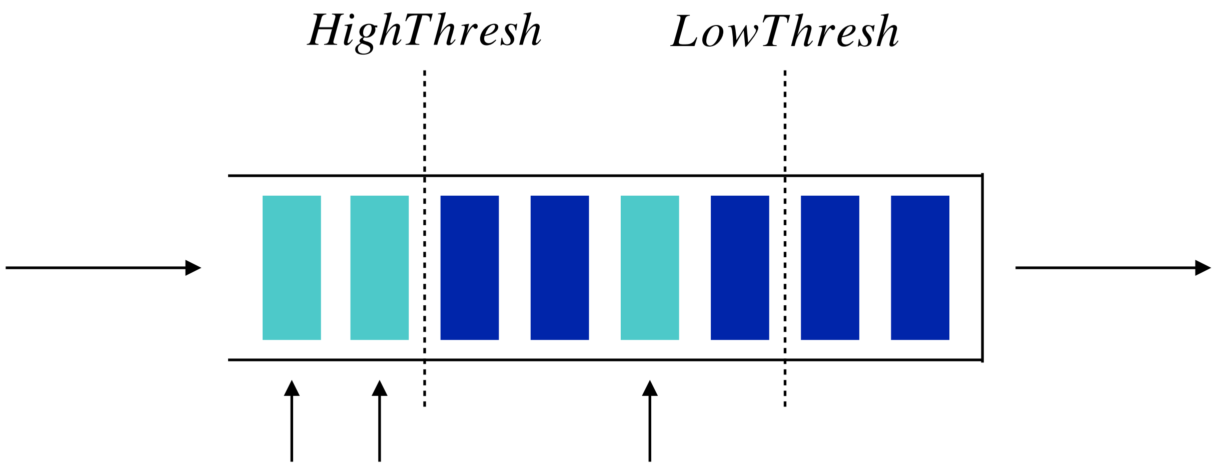
The dropped packet will serve as a signal for the sender to adjust its sending rate

AQM policy: random early detection (RED)



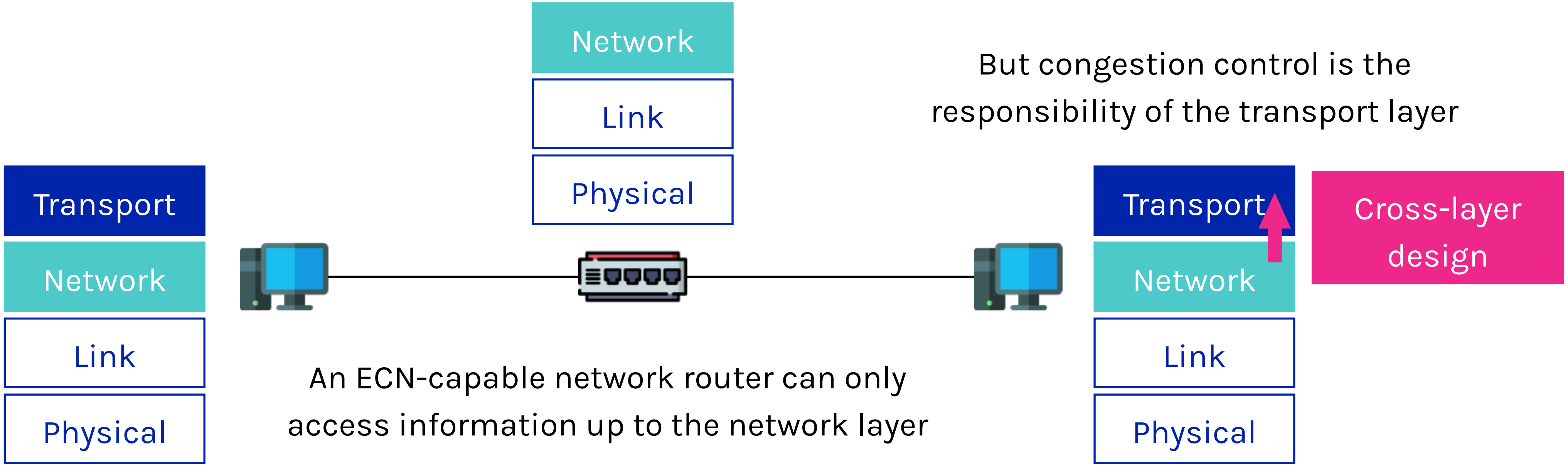
"Why should I drop perfectly good packets when I still have free buffer space?"

Explilict congestion notification (ECN)

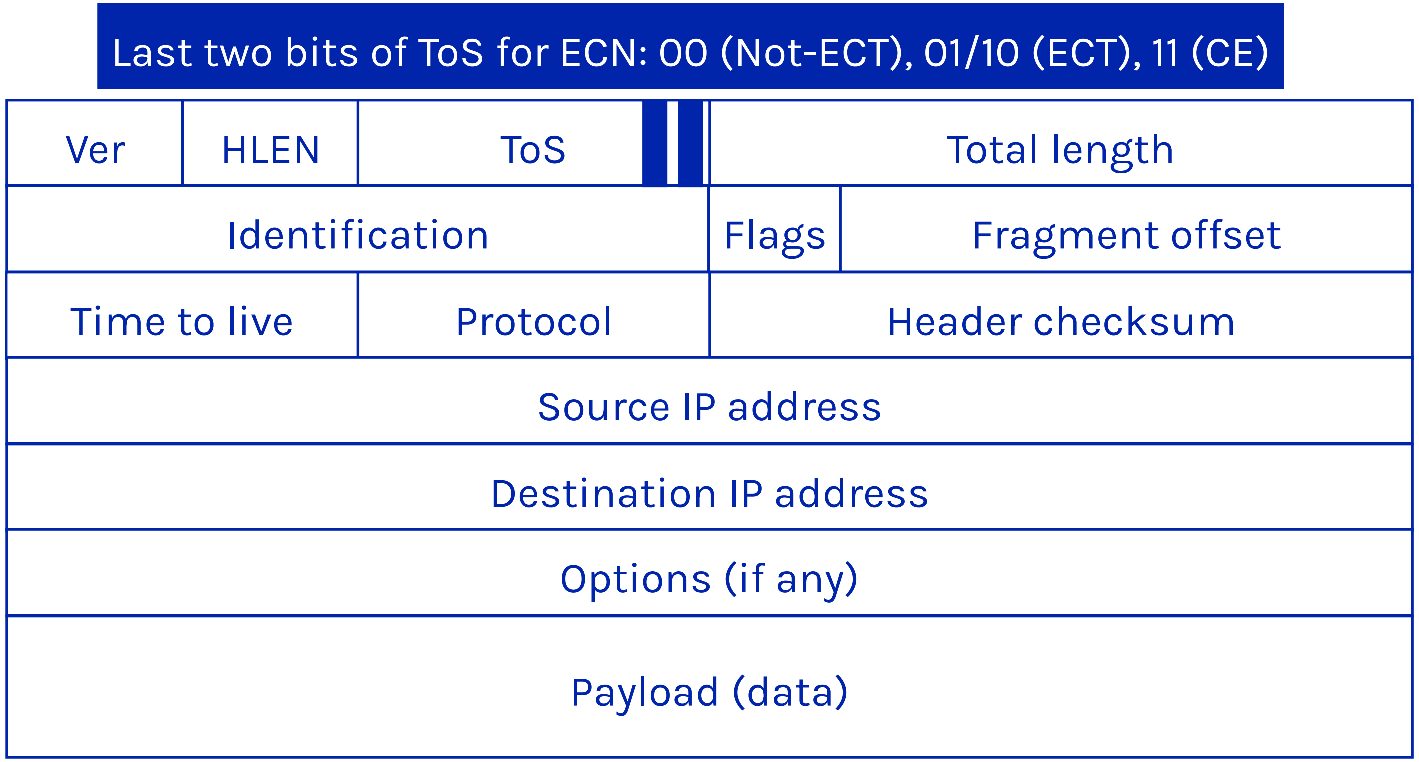


Instead of dropping, **mark** these packets and
notify the sender to reduce its sending rate

ECN mechanism

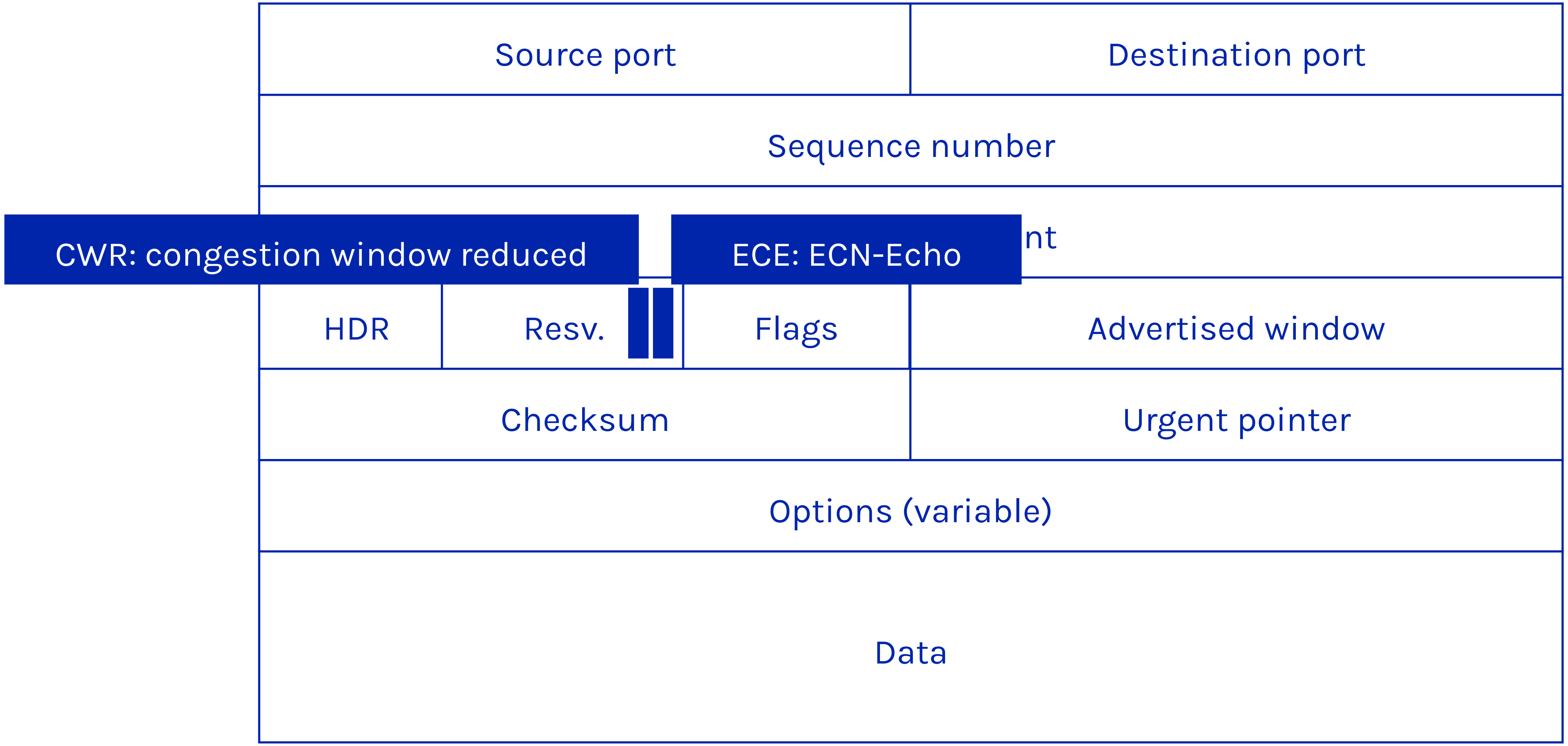


ECN mechanism: IP header

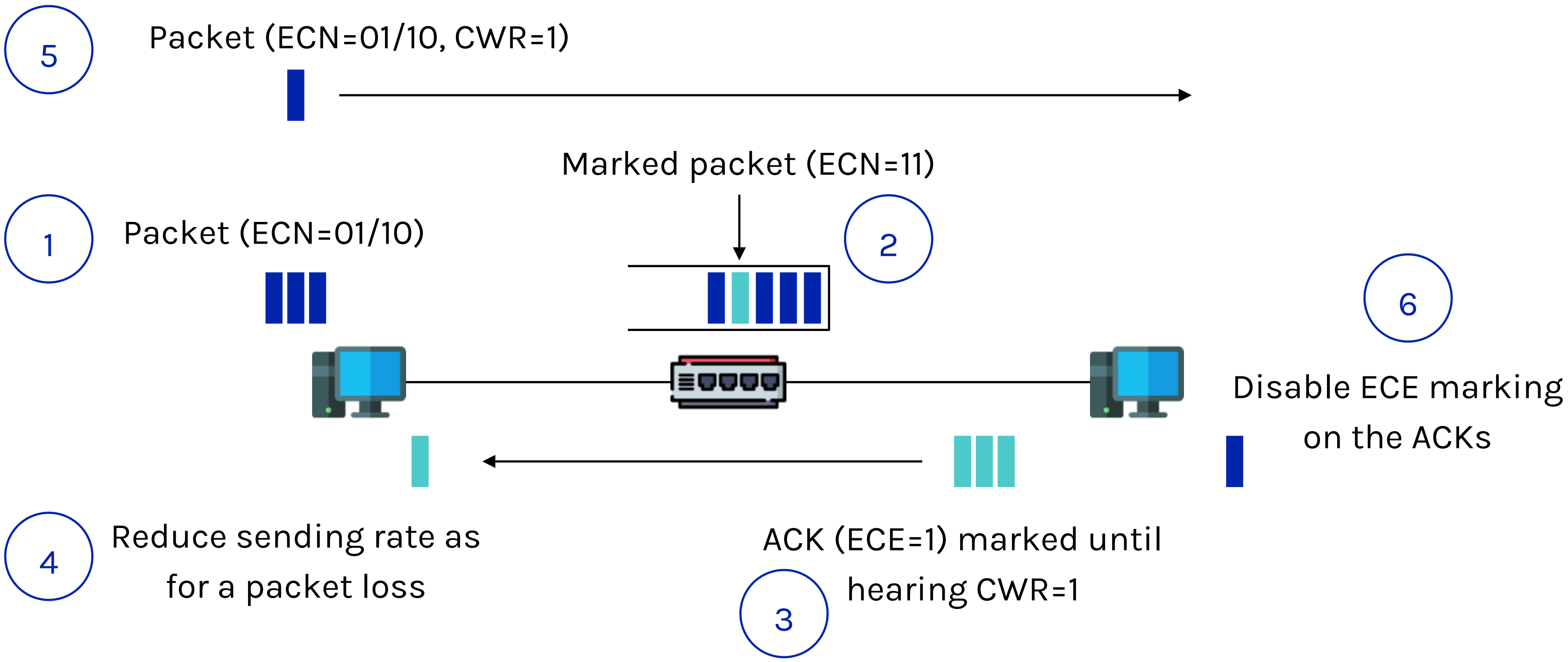


ECT: ECN-capable transport CE: Congestion experienced

ECN mechanism: TCP header



ECN example


5
Packet (ECN=01/10, CWR=1)
Marked packet (ECN=11)
1
Packet (ECN=01/10)
2
6
Disable ECE marking
on the ACKs
4
Reduce sending rate as
for a packet loss
ACK (ECE=1) marked until
hearing CWR=1
3

Summary

Network congestion

- Why congestion?
- Congestion quantification
- Congestion collapse of the Internet
- Congestion control goals
- Congestion window

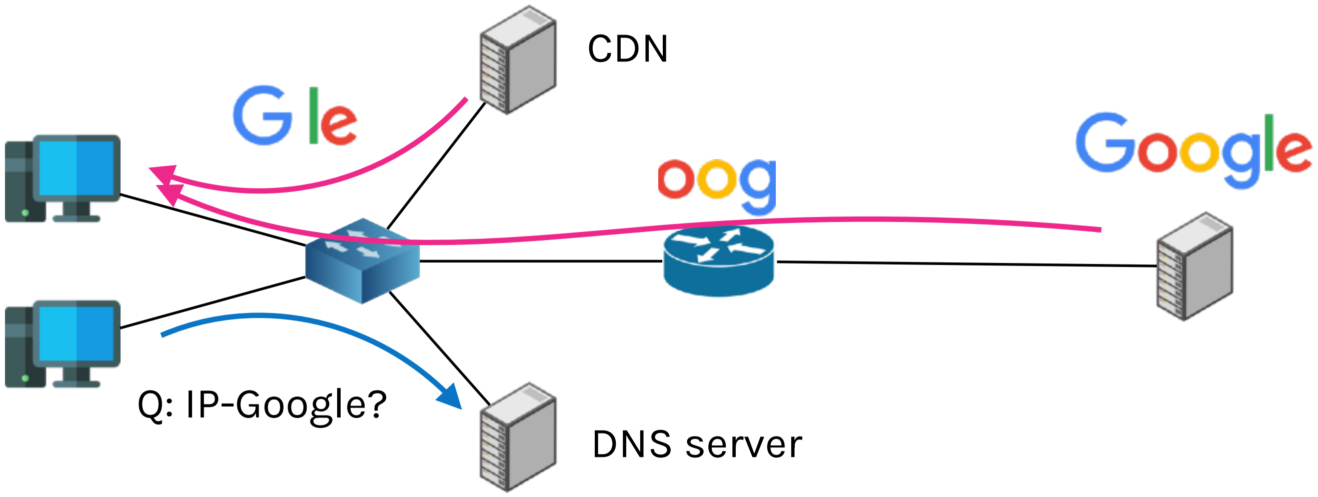
Congestion detection and control

- Congestion signals
- Bandwidth estimation (slow-start)
- Bandwidth adaptation
- Fairness
- AIMD
- Fast recovery

Explicit Congestion Notification

- AQM + RED
- ECN

Next time: application layer



How does the Web work?

Further reading material

Andrew S. Tanenbaum, David J. Wetherall. Computer Networks (5th edition).

- Section 6.5.10: TCP Congestion Control

Larry Peterson, Bruce Davie. Computer Networks: A Systems Approach.

- Chapter 6: Congestion Control