Network-assisted resource sharing

CSE 561, Winter 2021

Ratul Mahajan

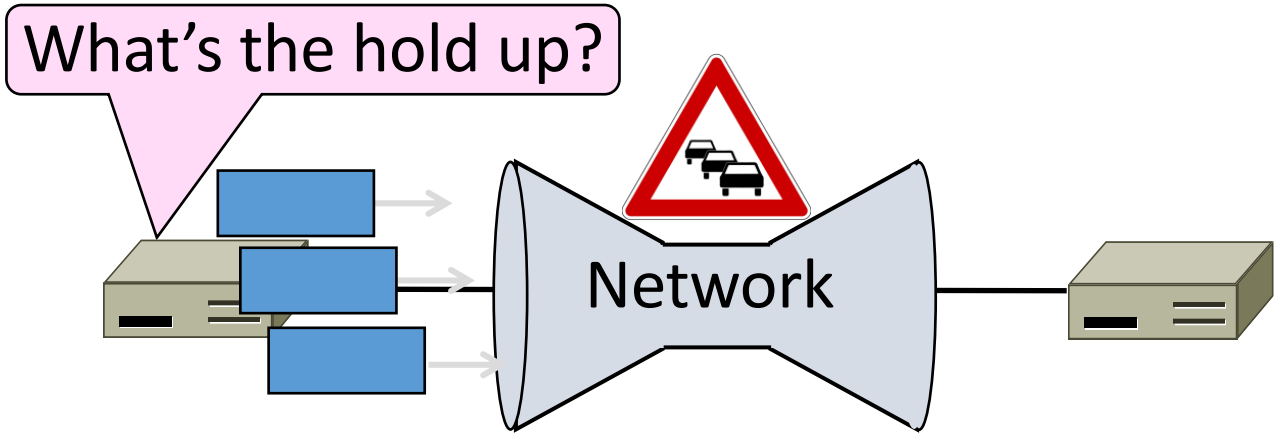
What we read

3 different switch mechanisms to reduce congestion, improve fairness

- Fair queueing
- Credit-based flow control
- Random early drop

Problem: Network congestion

A “traffic jam” in the network



Congestion Collapse in the 1980s

Early TCP used fixed size window (e.g., 8 packets)

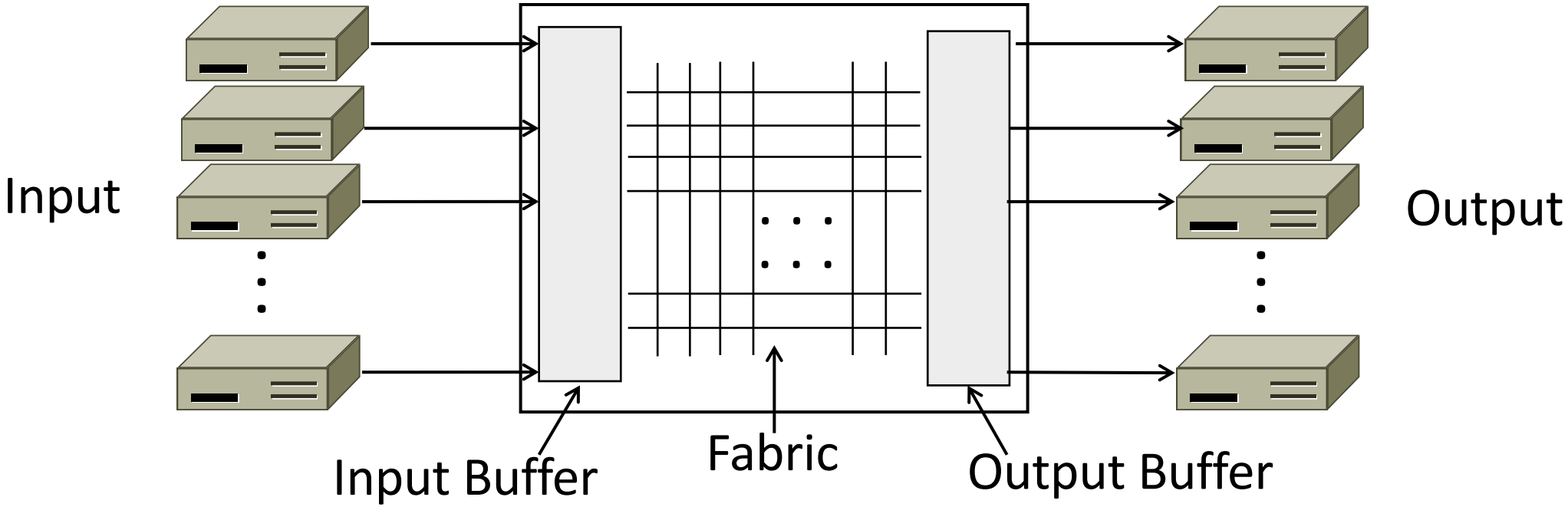
- Initially fine for reliability

But something happened as the network grew

- Links stayed busy but transfer rates fell by orders of magnitude!

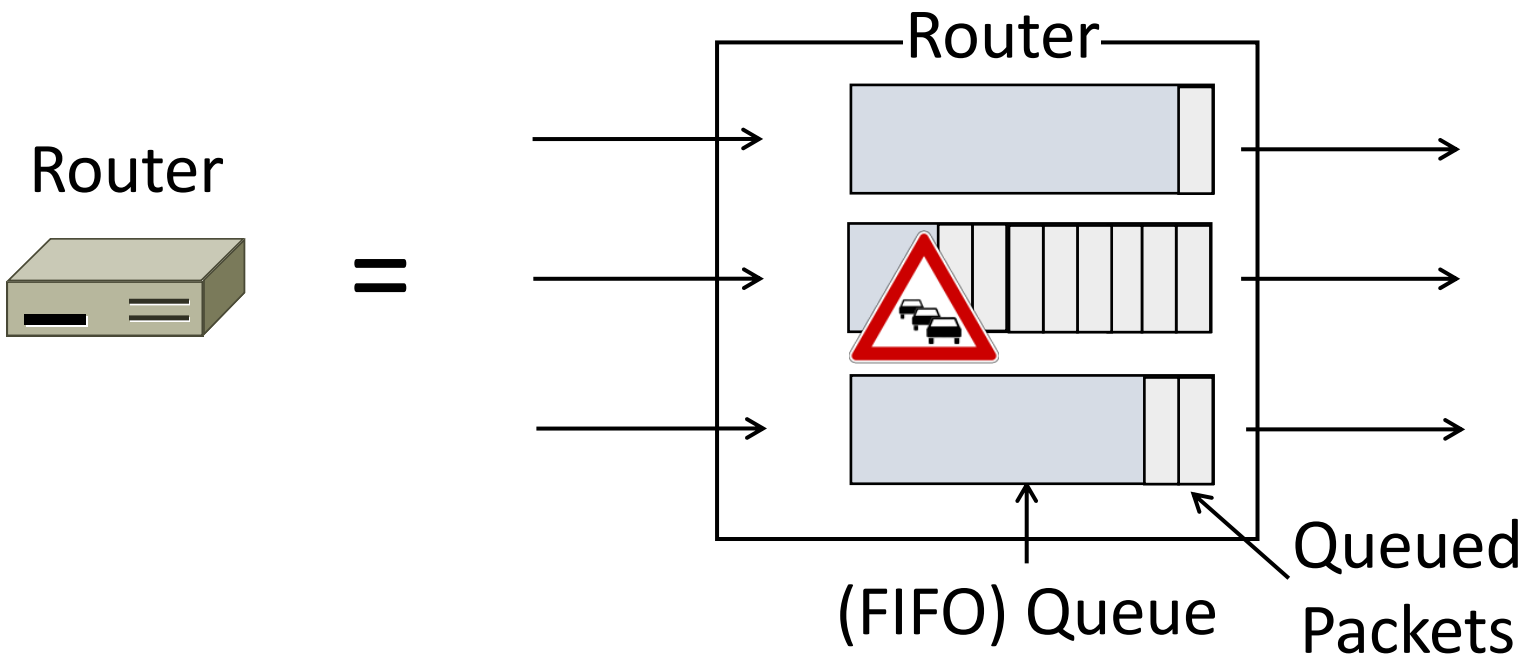
Nature of Congestion

Routers/switches have internal buffering



Nature of Congestion (2)

Simplified view of per port output queues



Notes on congestion

Buffers help absorb bursts when input > output rate

- Buffer sizes need to be carefully picked

But if input > output rate persists, queue will overflow

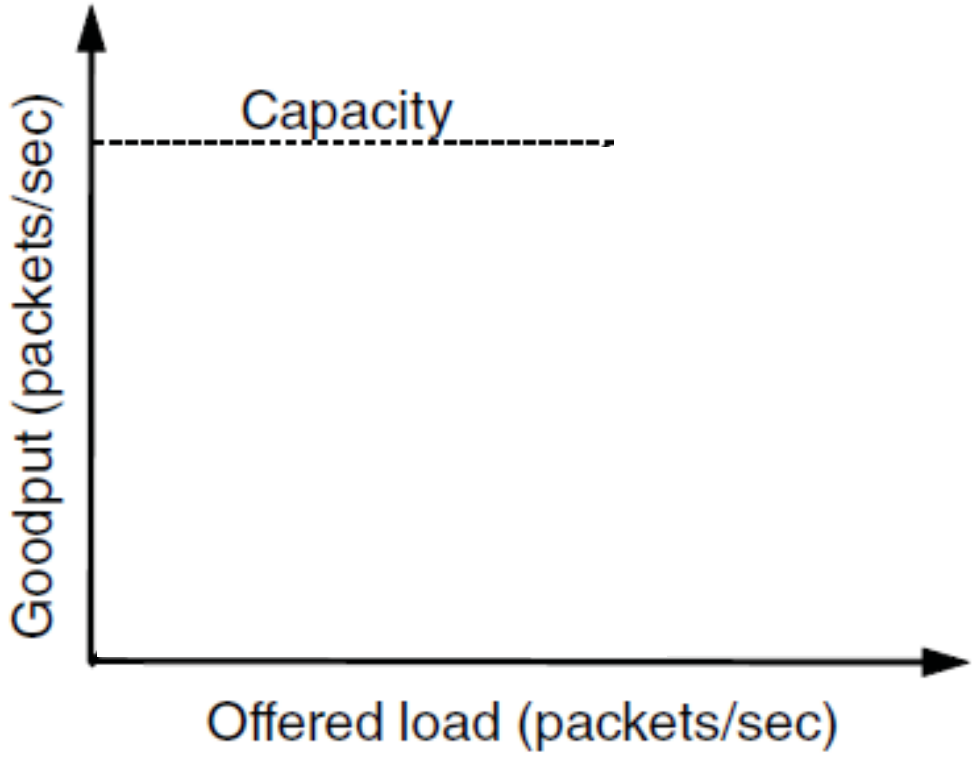
→ This is congestion

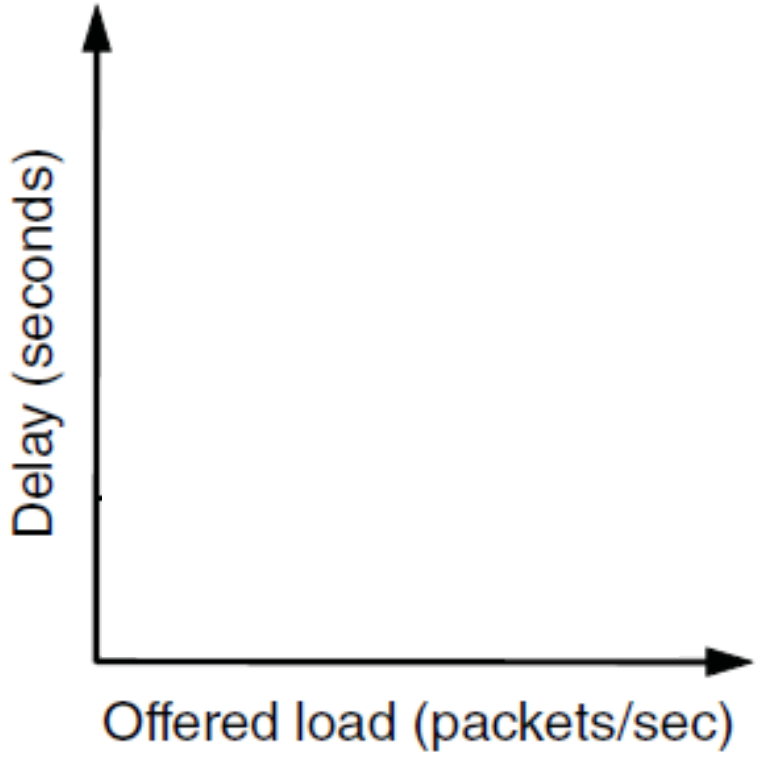
Congestion is a function of the traffic patterns

- Can occur even if every link has the same capacity

Effects of Congestion

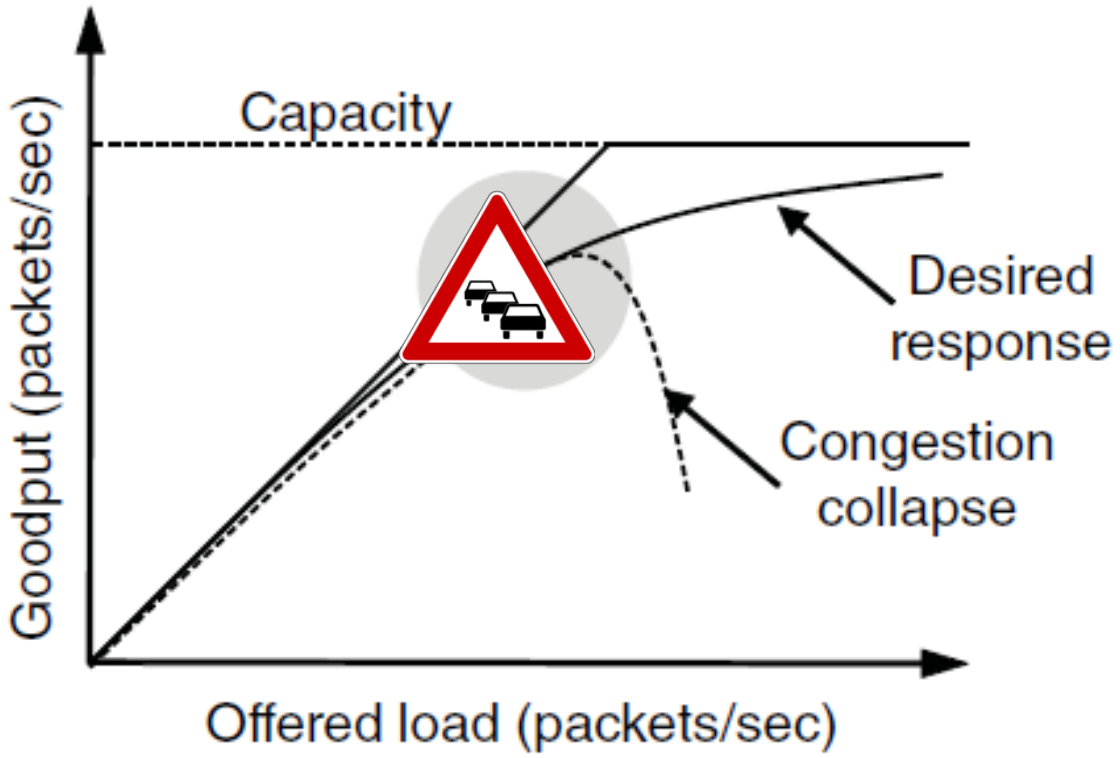
Performance as we increase load

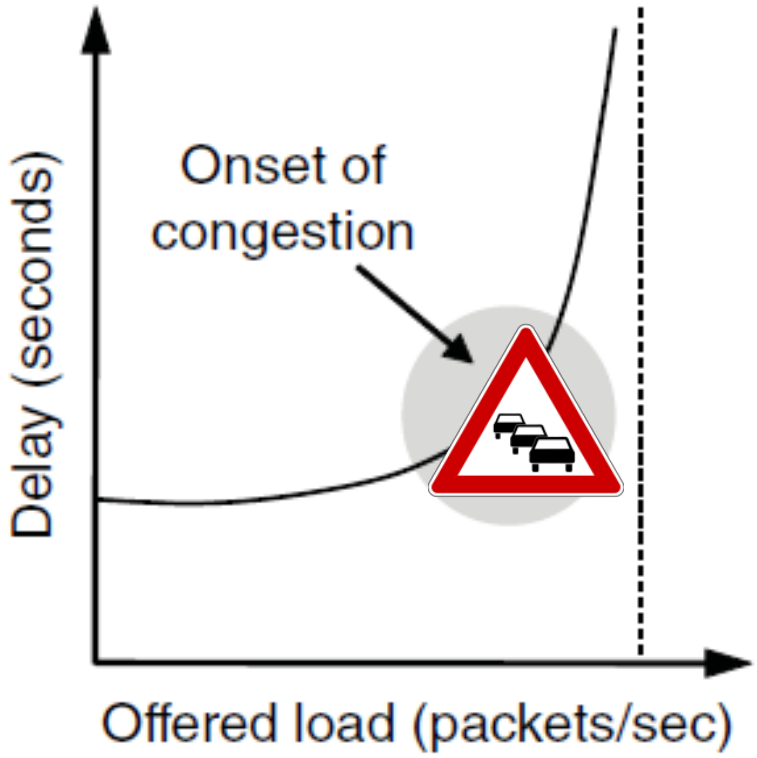




Effects of Congestion

Performance as we increase load





Effects of Congestion (2)

Offered load rises → congestion as queues fill

- Delay and loss rise sharply with load
- Throughput < load (packet loss)
- Goodput << throughput (spurious retransmissions)

None of the above is good!

- Want network performance just before congestion

Goals of resource allocation

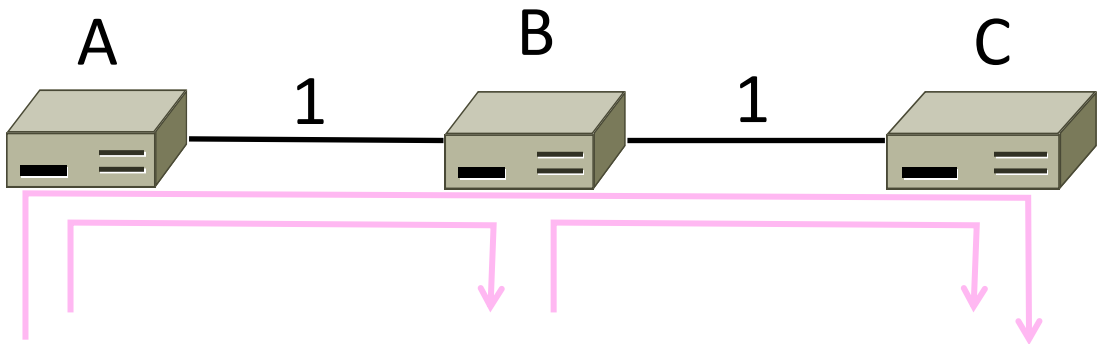
1. Efficient use of network resources
   - Low latency

2. Fairness
   - Protection from bad actors
   - Max-min fairness

Efficiency vs. Fairness

Cannot always have both!

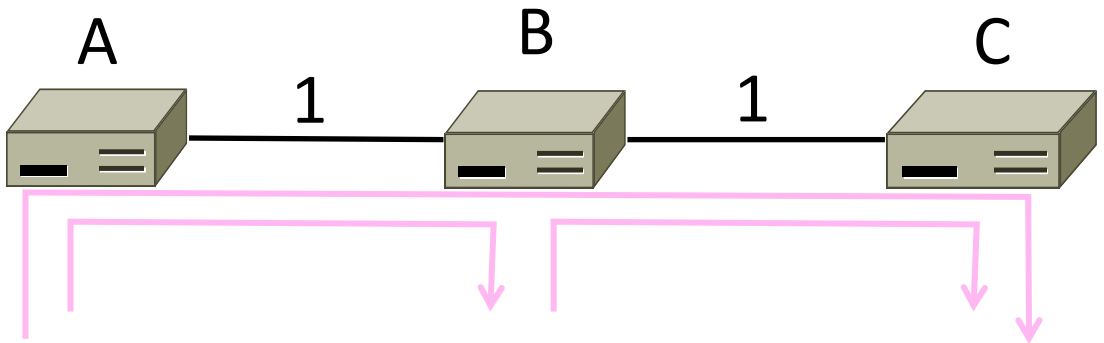
- Example network with traffic: A→B, B→C and A→C
- How much traffic can we carry?



Efficiency vs. Fairness (2)

If we care about fairness:

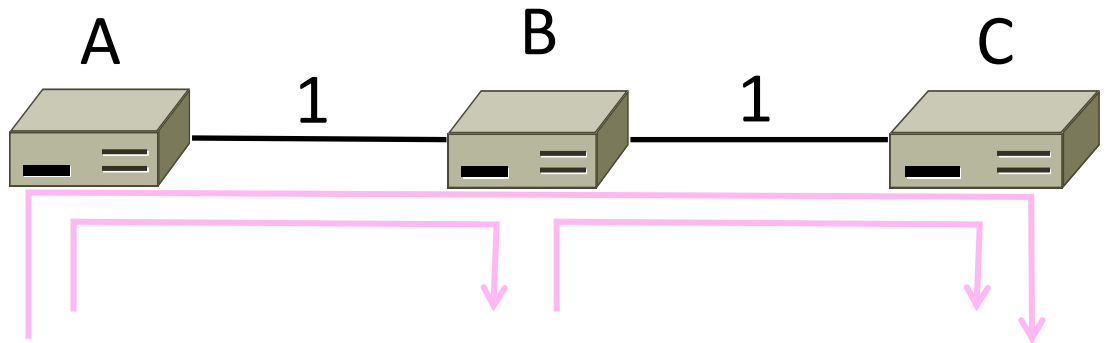
- Equal per flow: A→B: ½ unit, B→C: ½, and A→C, ½
- Total traffic carried is 1 ½ units



Efficiency vs. Fairness (3)

If we care about efficiency:

- Maximize traffic: A→B: 1 unit, B→C: 1, and A→C, 0
- Total traffic rises to 2 units!



The Slippery Notion of Fairness

Why is “equal per flow” fair anyway?

- A→C uses more network resources than A→B or B→C
- Host A sends two flows, B sends one

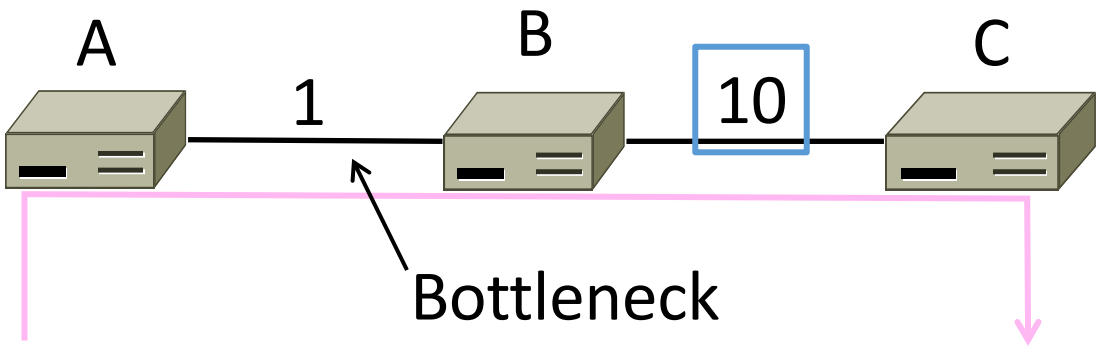
Often not productive to seek exact fairness

- More important to avoid starvation
  - A node that cannot use any bandwidth
- “Equal per flow” is good enough

Generalizing “Equal per Flow”

Bottleneck for a flow is the limiting link

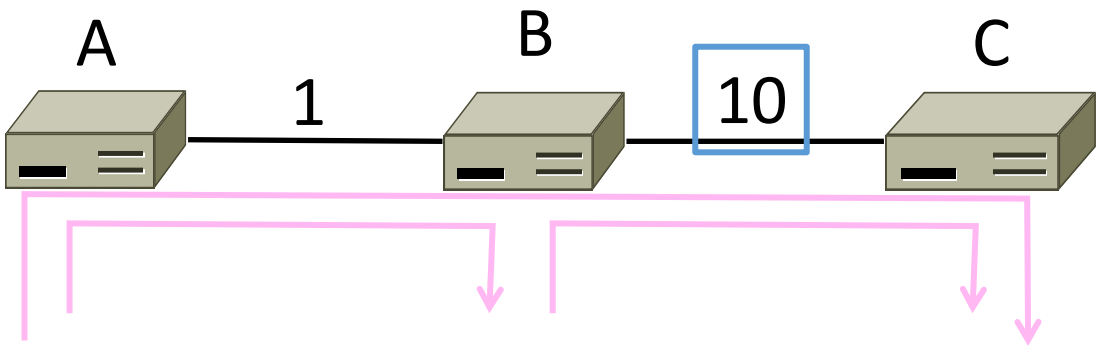
- Where congestion occurs for the flow
- For A→C, link A–B is the bottleneck



Generalizing “Equal per Flow” (2)

Flows may have different bottlenecks

- For A→C, link A–B is the bottleneck
- For B→C, link B–C is the bottleneck
- Can no longer divide links equally …



Max-Min Fairness

Intuitively, flows bottlenecked on a link get an equal share of that link

Max-min fair allocation:

- Increasing the rate of one flow will decrease the rate of a smaller flow
- This “maximizes the minimum” flow

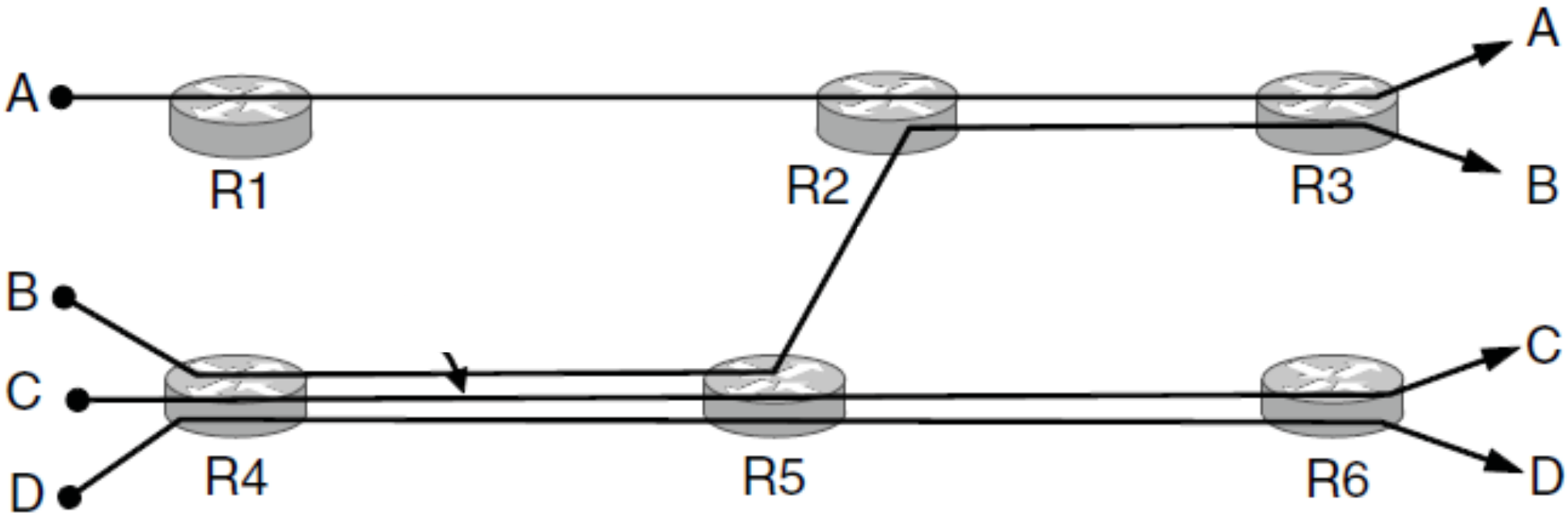
Max-Min Fairness (2)

To find it, imagine “pouring water into the network”

1. Start with all flows at rate 0
2. Increase the flows until there is a new bottleneck in the network
3. Hold fixed the rate of the flows that are bottlenecked
4. Go to step 2 for any remaining flows

Max-Min Example

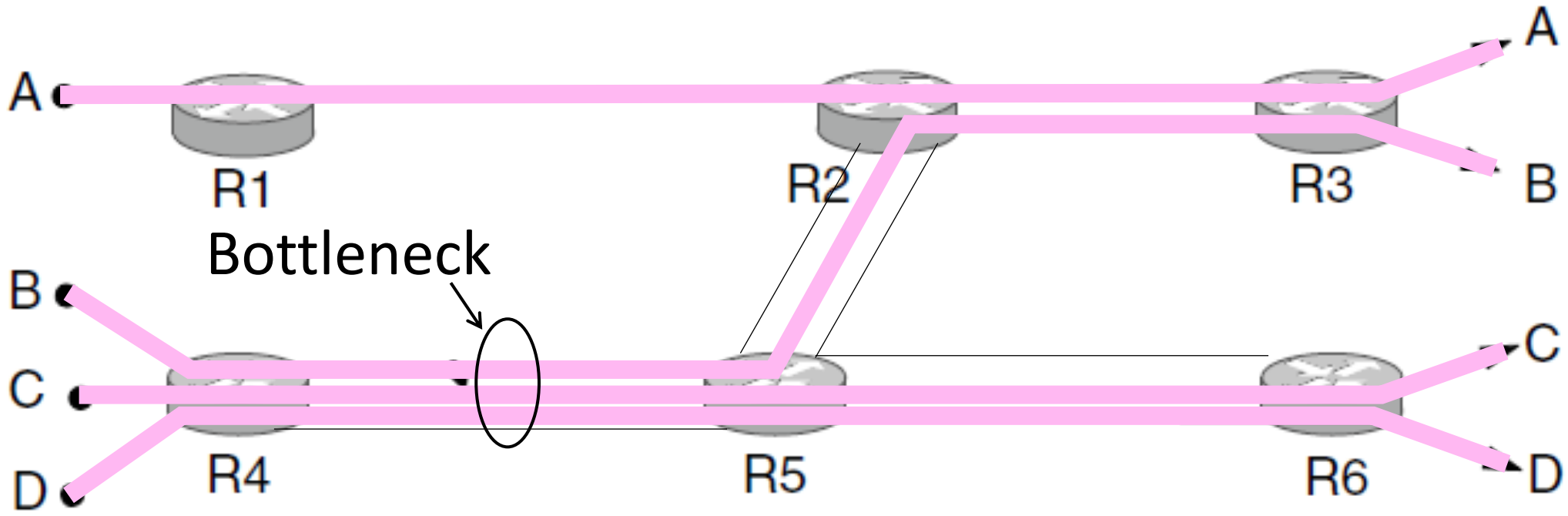
- Example: network with 4 flows, link bandwidth = 1
  - What is the max-min fair allocation?



Max-Min Example (2)

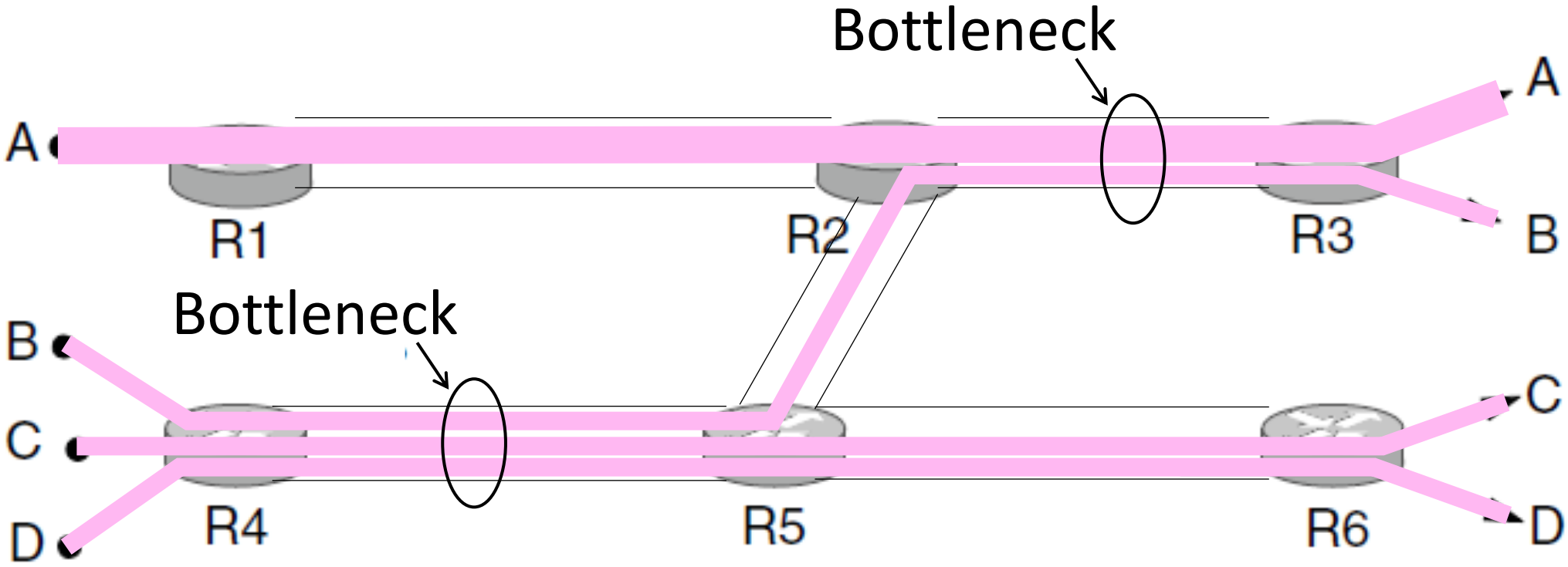
When rate=1/3, flows B, C, and D bottleneck R4—R5

- Fix B, C, and D, continue to increase A



Max-Min Example (3)

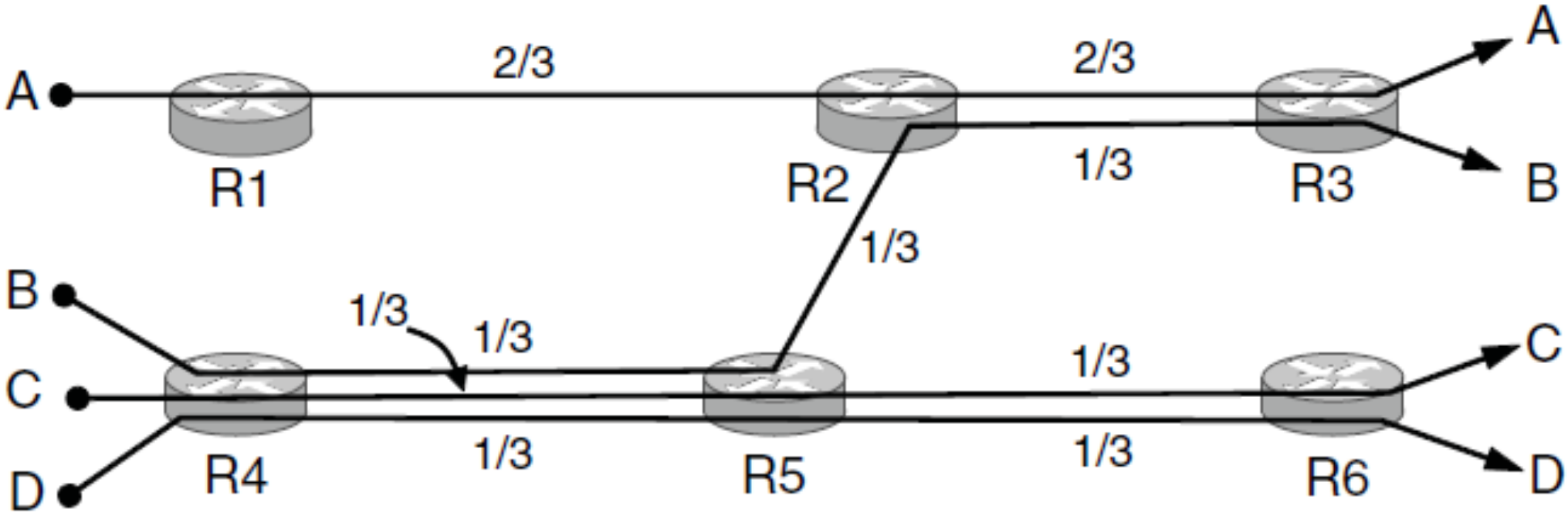
When rate=2/3, flow A bottlenecks R2—R3. Done.



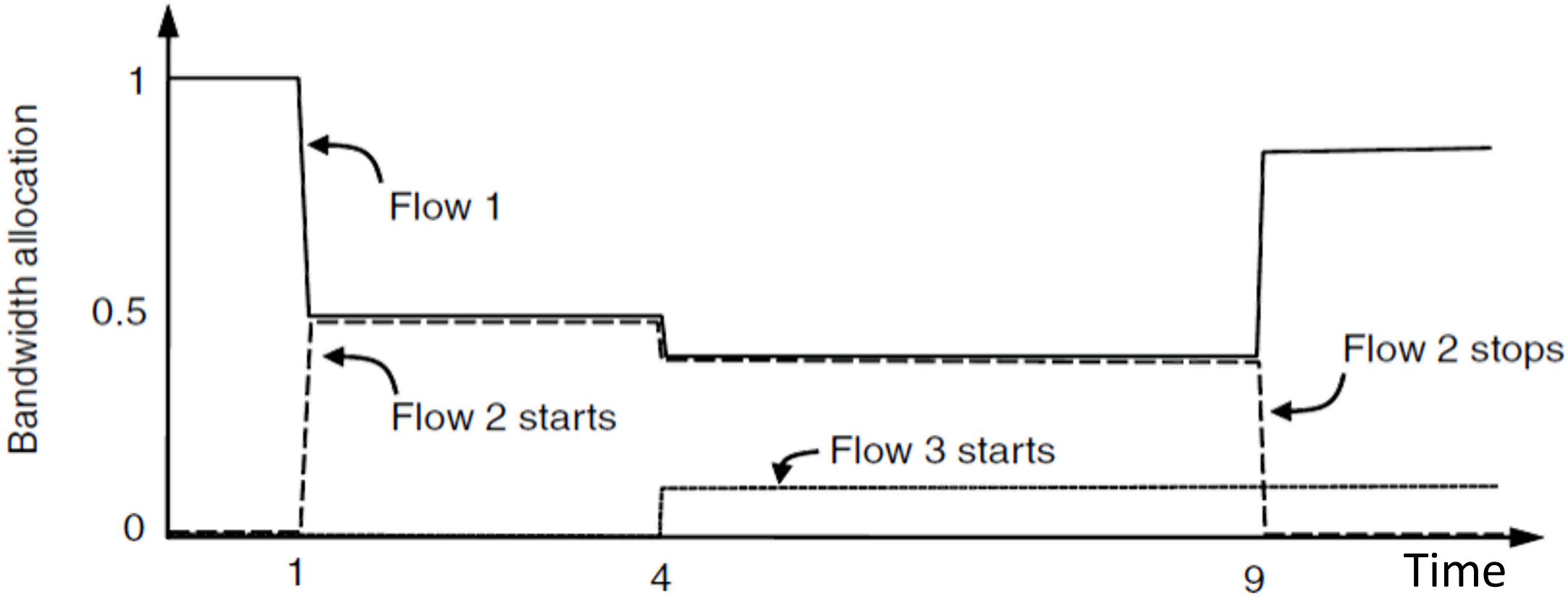
Max-Min Example (4)

End with A=2/3, B, C, D=1/3, and R2—R3, R4—R5 full

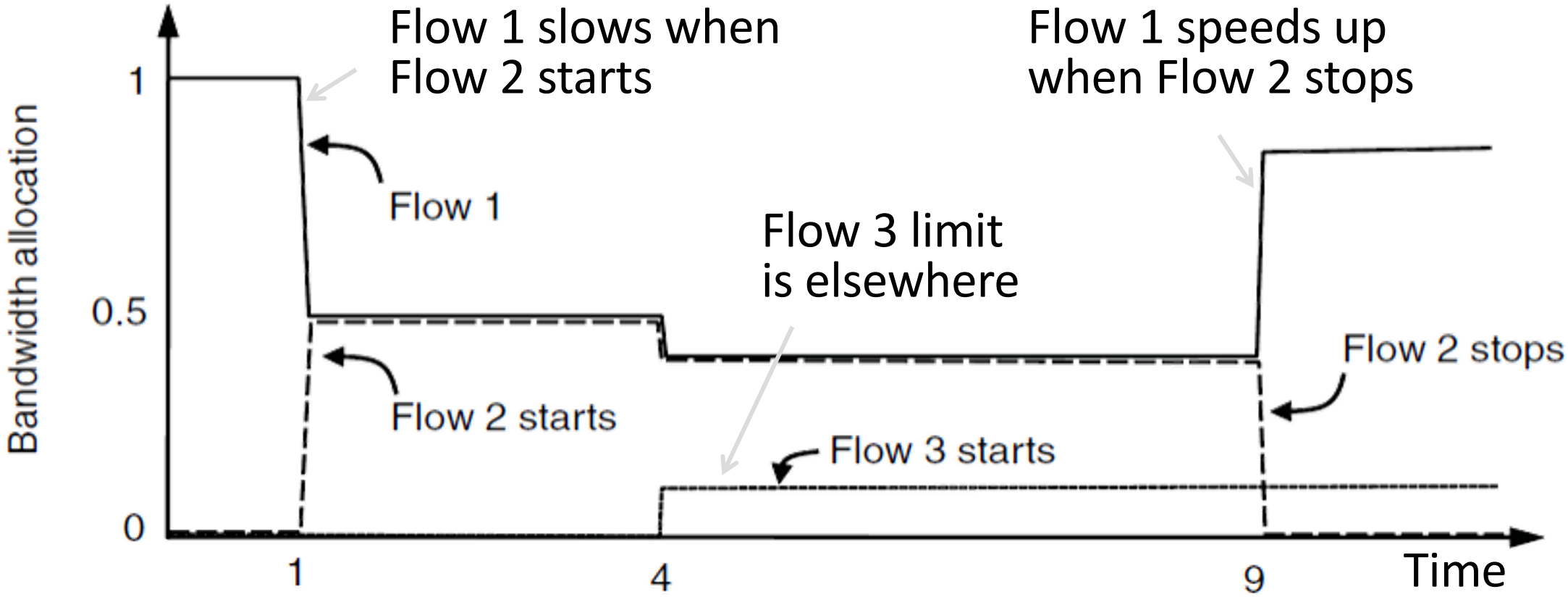
- Other links have extra capacity that can't be used



Adapting over Time



Adapting over Time



Three ways to allocate network resources

1. Pick paths intelligently
   - Routing and traffic engineering

2. Get hosts to be smart about how much they send → later
   - End-host congestion control (TCP)

3. Get routers involved → today
   - Send feedback to end hosts
   - Protect some flows from others

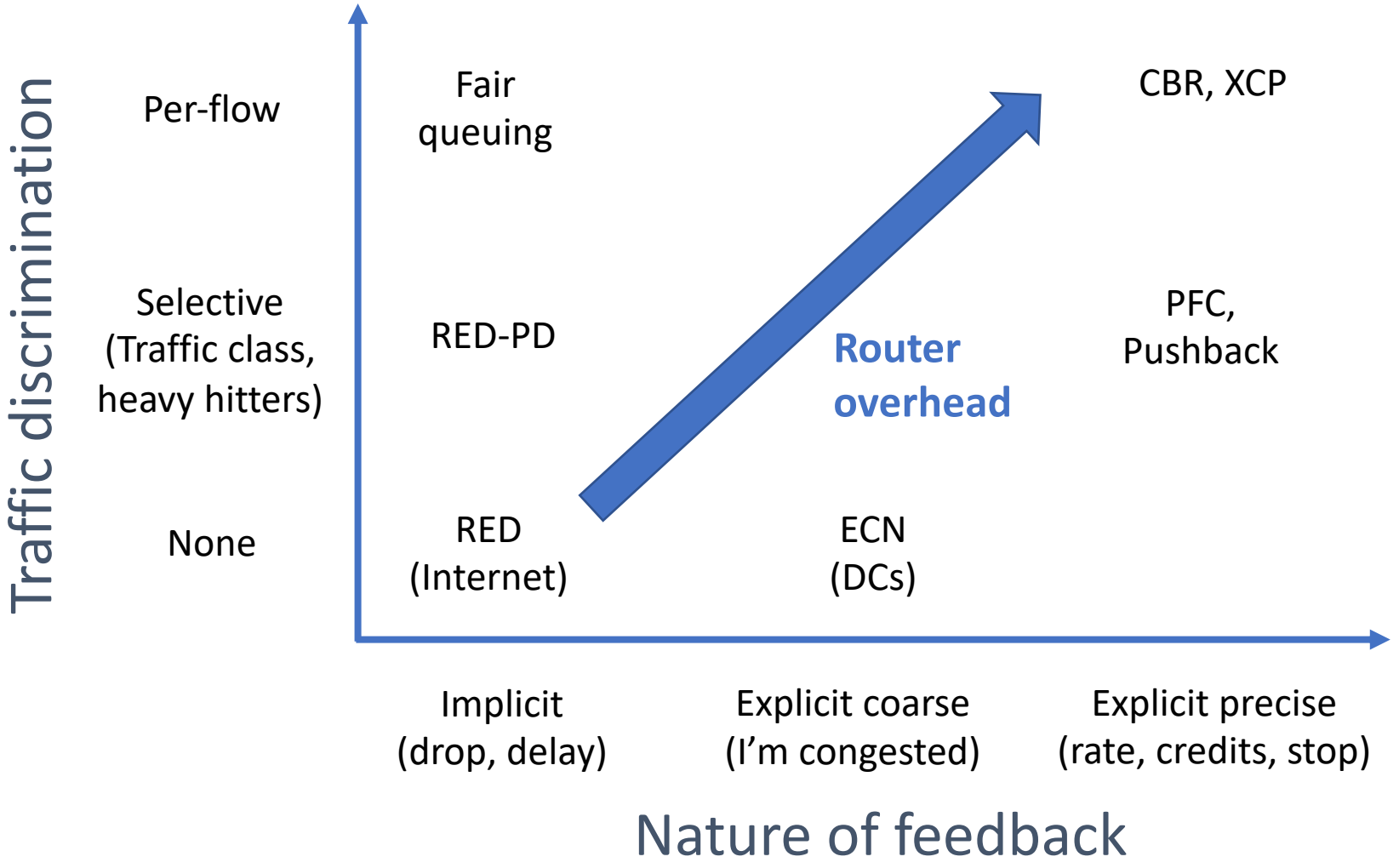
Case for router-assist

Definitive view of congestion

- Loss: End hosts cannot distinguish between congestion and corruption
- Delay: End hosts cannot distinguish between congestion and long paths

Only way to protect some sources from others

Space of router assisted resource allocation



Packet handling

- Buffer management
  - DropTail, RED, ..
- Scheduling
  - FIFO, RR, …

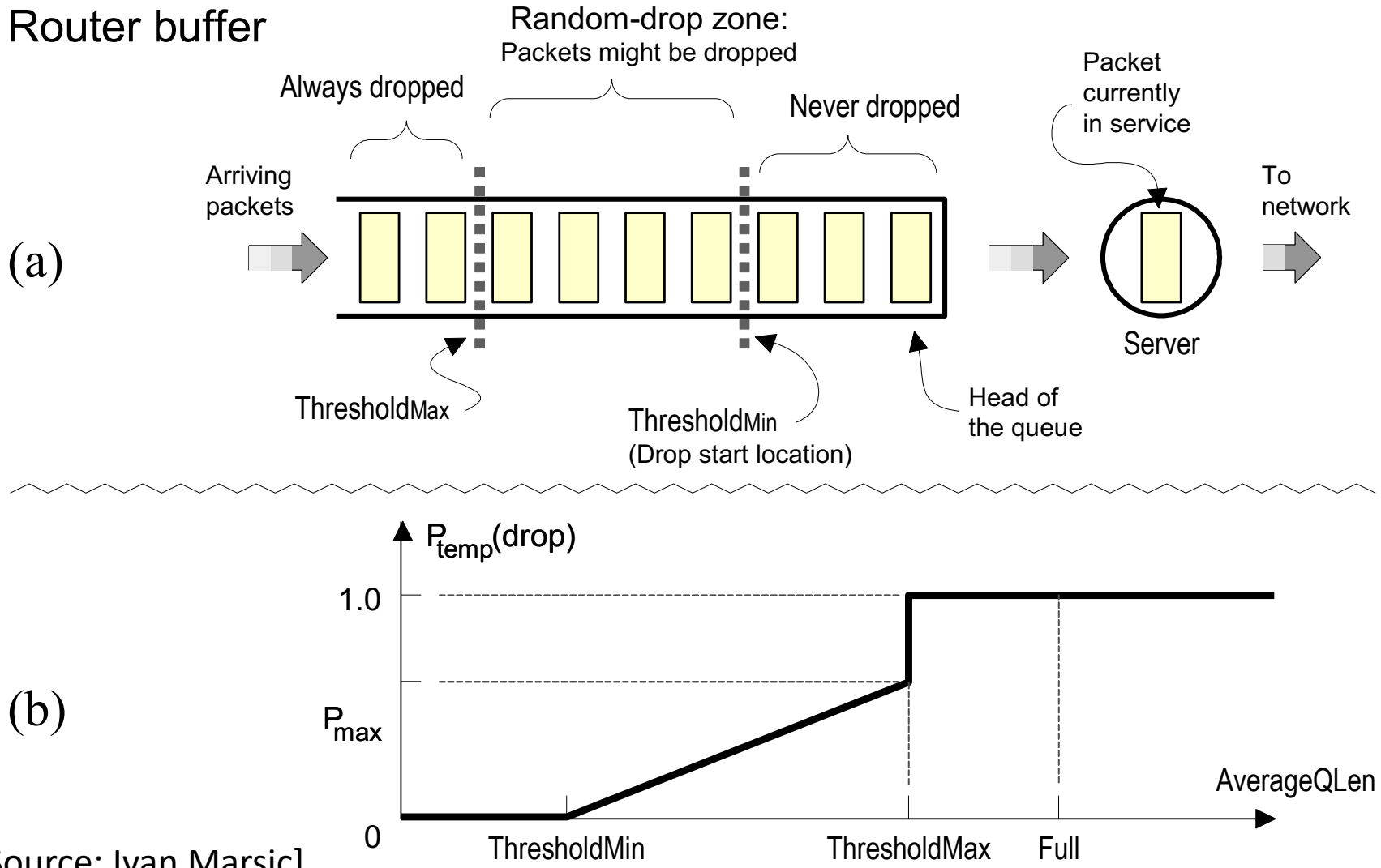
RED: Random Early Detection

Random Early Detection (RED)

Buffer management: Drop early and probabilistically

Scheduling: FIFO

RED buffer management



[Source: Ivan Marsic]

ECN version: Mark instead of dropping

RED notes

Avoids “boom and bust” cycles

Controls queue lengths to avoid getting into high-delay territory

No protection from misbehaving sources

RED retrospective

Implemented widely in routers

But rarely turned on

- Parameters were hard to tune
  - Optimal is a complex function of number of flows, RTTs, connection sizes, ..

Early feedback ideas are seeing a resurgence in DCs now

Fair queueing

[Some slides from Dave Andersen]

Fair queueing

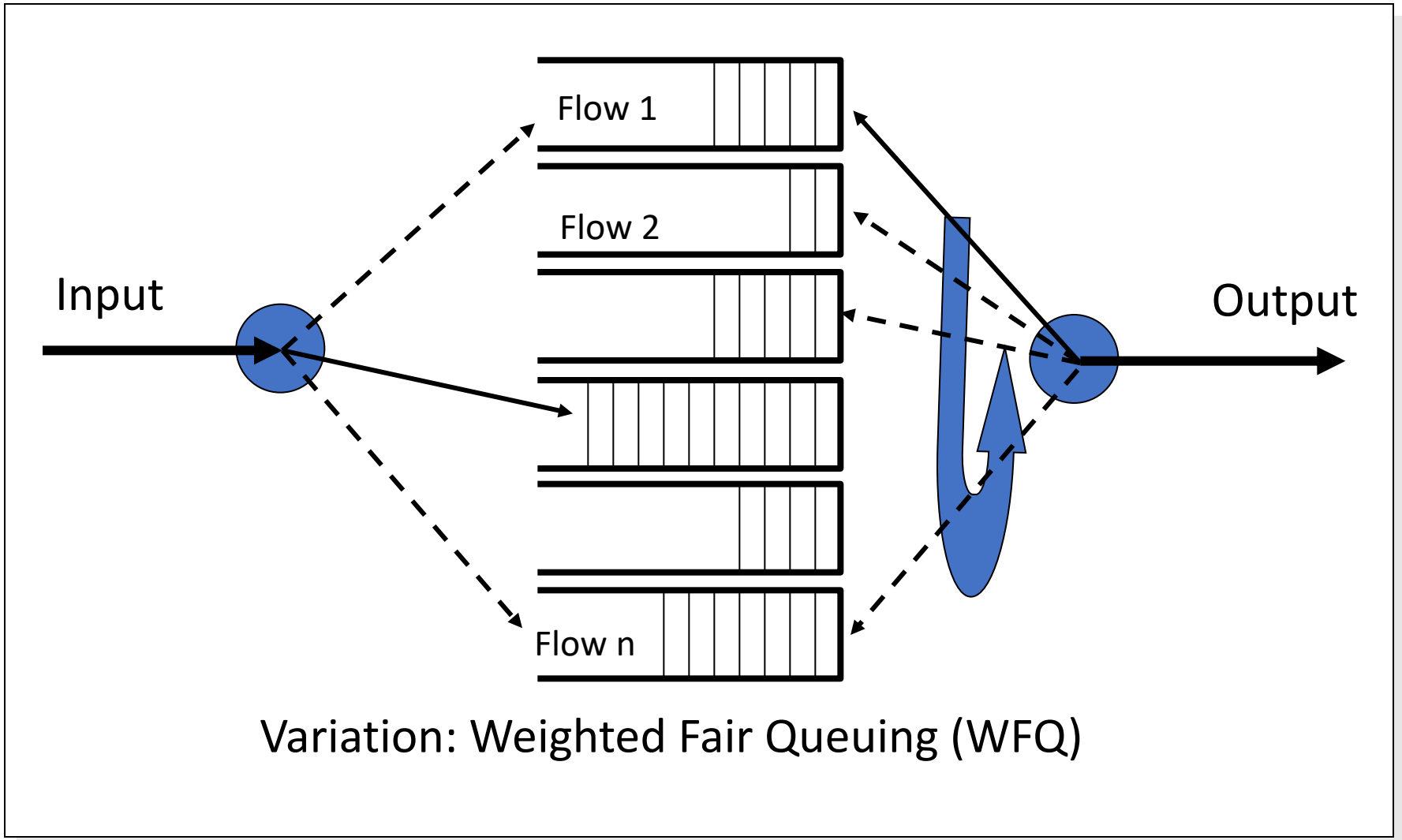
Buffer management:

- Per-flow queues and drop flow with the longest queue

Scheduling

- Round robin

FQ Illustration



Bit-by-bit RR

Multiple flows: clock ticks when a bit from *all* active flows is transmitted → a "round"

- dR/dt (the rate at which the round #increases) is *variable* = μ / N
  - μ = #bits/sec router can send
  - N = # active flow

- Why count this way? # of rounds to send a packet is *independent* of number of active flows. Useful way of viewing things…

Bit-by-bit RR

Packet arrives in queue Q:

- It's the *i*th packet in the queue
- It's p_i^q bits long
- When does it start being transmitted?
  - If q empty, immediately: R(t)
  - Else, just after prior pkt finishes: F_{i-1}^q
  - S_i^q = max(R(t), F_{i-1}^q)
- When does it complete?
  - S_i^q + p_i^q (p_i^q rounds later…)
- Can compute the finish *round* of every packet in the queue. (Even at the point when the packet is enqueued).
  - Note that we don't know the finish *time*, just the round #.

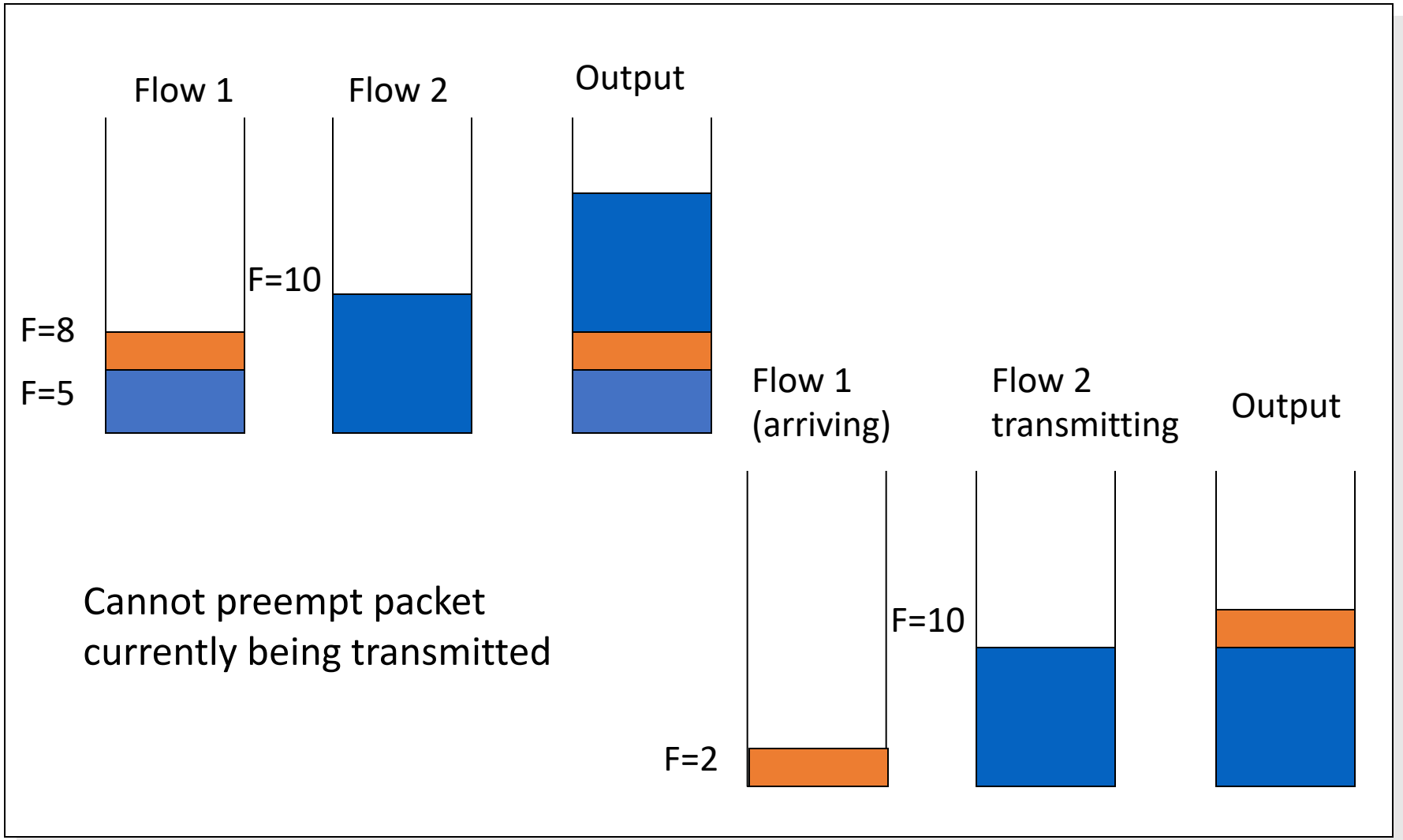
Packet-based Fair Queueing

Simple: Send the packet with the smallest finishing round #.

Approximates bit-by-bit RR

- Why isn’t it exact? Preemption!

Bit-by-bit RR Example



FQ notes

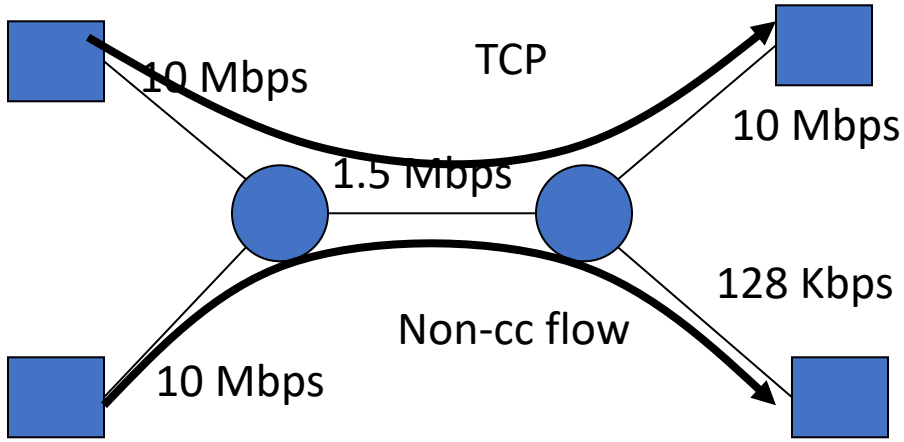
A lighter-weight version DRR (deficit RR) implemented in some switches but many switches don’t implement it

Use is rare (even rarer that RED)

- Flow definition is a challenge
- Considered too expensive

One idea for reducing cost: CSFQ (core stateless FQ)

Does FQ prevent congestion collapse?



Example from Floyd and Fall, 1999

No. Still need end-to-end congestion control

Credit-based flow control

Credit-based flow control

Buffer management

- Per flow (VC)

Scheduling

- Round robin

Same decisions as FQ but a very different mechanism

CBFC vs FQ

CBFC sends explicit, detailed feedback upstream

- Number of packets (“cells”) upstream can send

Other possible feedbacks

- Rate
- Pause

CBFC

Receivers

- Tell senders how many packets they can send based on buffer allocation
- Called credits

Senders

- Transmit only when they have non-zero credits
- Decrement credit with each transmission

Most complexity hidden inside managing buffer allocation

Flow control vs. congestion control

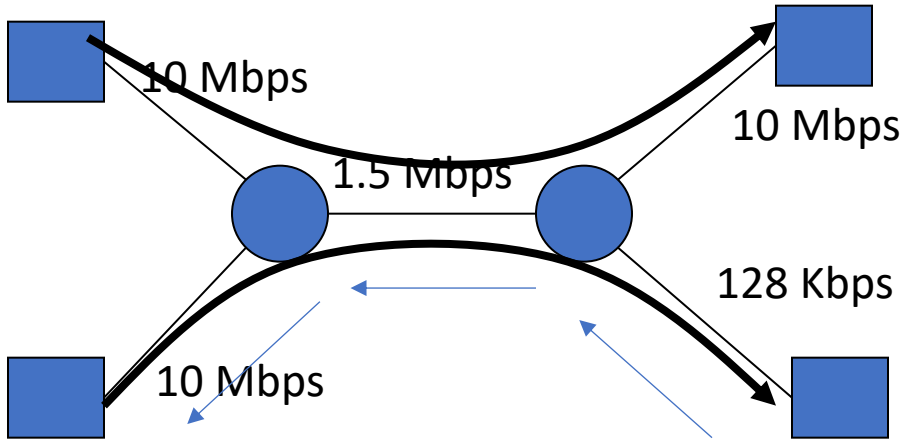
Flow control

- Match transmission rate of the sender to what the receiver can bear
- Can be hop-by-hop or end-to-end

Congestion control

- Prevent congestion in the middle of the network
- A worry for packet-switched networks

Backpressure to source helps avoid congestion collapse



Credit-based flow control notes

Not implement in a network anywhere (to my knowledge)

PFC is the closest deployed mechanism

- Ask a class of traffic to stop when they run the risk of overflowing buffers

Your thoughts

Can CBFC handle bursty traffic?

Why are timers the Bermuda triangle?

Next class

First guest lecture!

Kurtis Heimerl on Cellular Networks