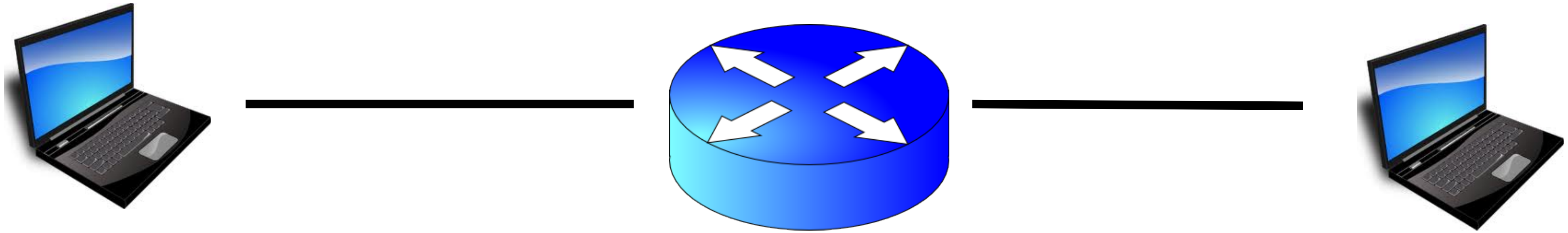
Transport

Lecture 4, Computer Networks (198:552)

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Network Edge: Application guarantees

- Endpoints should provide guarantees to applications



- Transport software on the endpoint is in charge of implementing guarantees on top of an unreliable network

Modularity through layering

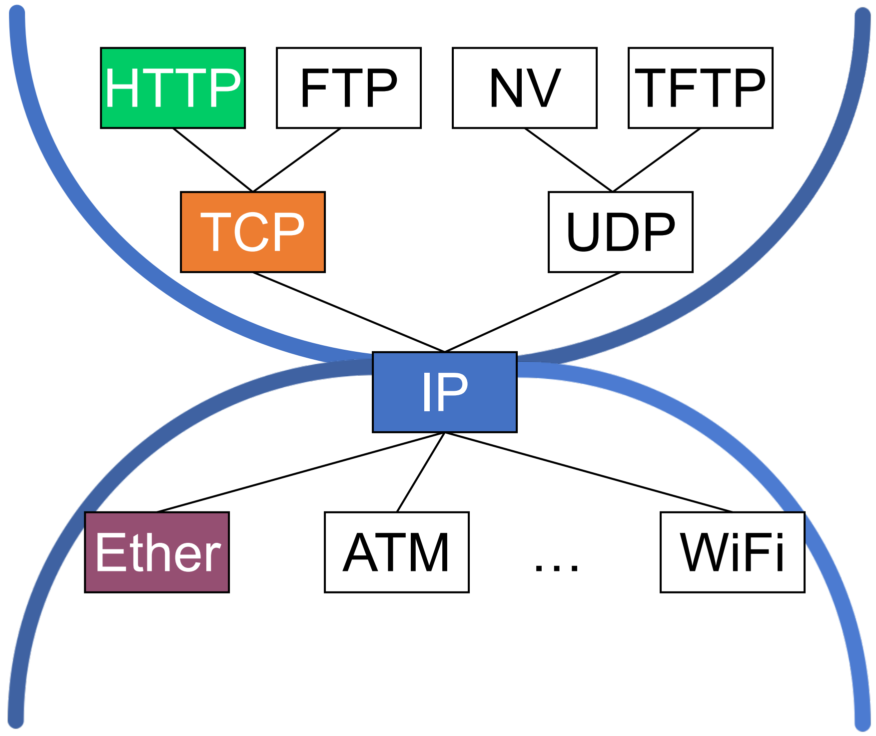
Protocols “stacked” in endpoint and router software/hardware

Apps: useful user-level functions

Transport: provide guarantees to apps

Network: best-effort global pkt delivery

Link: best-effort local pkt delivery

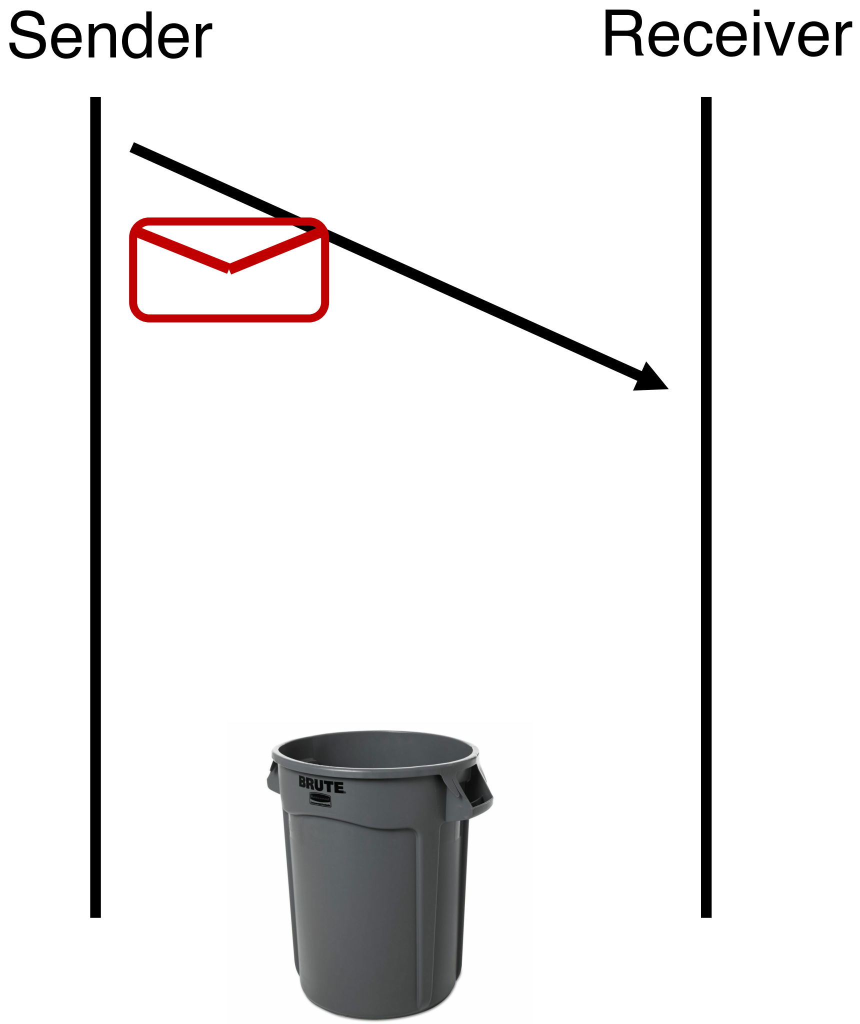


Transmission Control Protocol (TCP)

- Multiplexing/demultiplexing
  - Determine which conversation a given packet belongs to
  - All transports need to do this
- Reliability and flow control
  - Ensure that data sent is delivered to the receiver application
- Ordered delivery
  - Ensure bits pushed by sender arrive at receiver app in order
  - Q: why would packets ever be received out of order?
- Congestion control
  - Ensure that data sent doesn't overwhelm network resources
  - Q: which network resource?

Reliable data delivery

Packet loss



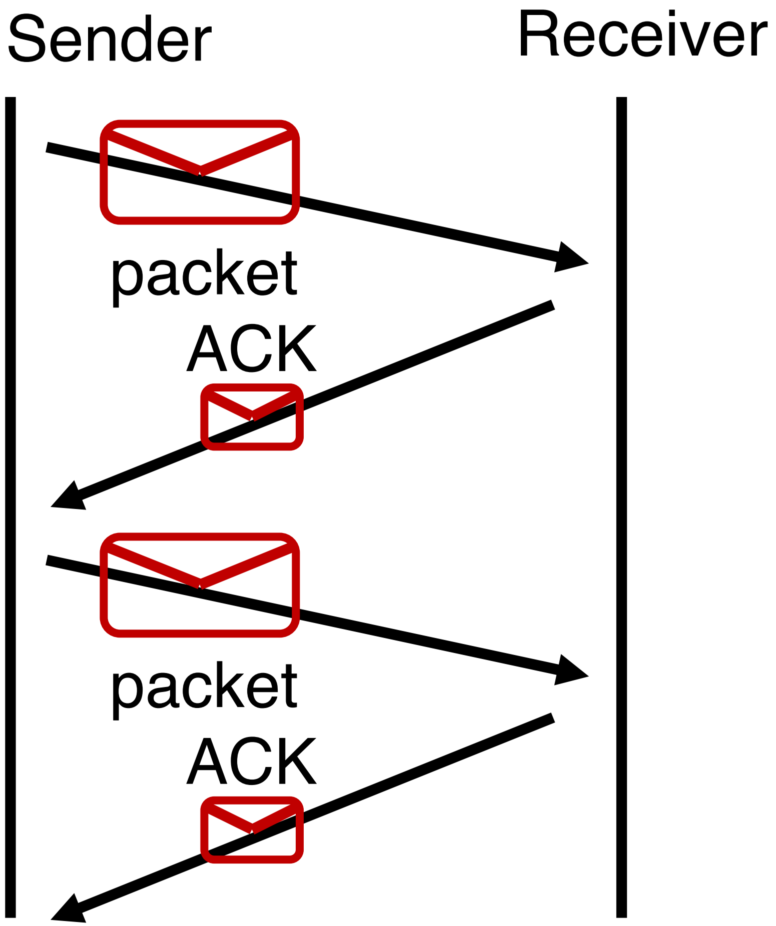
- How might a sender and receiver ensure that data is delivered reliably (despite some packets being lost)?
- TCP uses two mechanisms

Coping with packet loss: (1) ACK

- Key idea: Receiver returns an acknowledgment (ACK) per packet sent

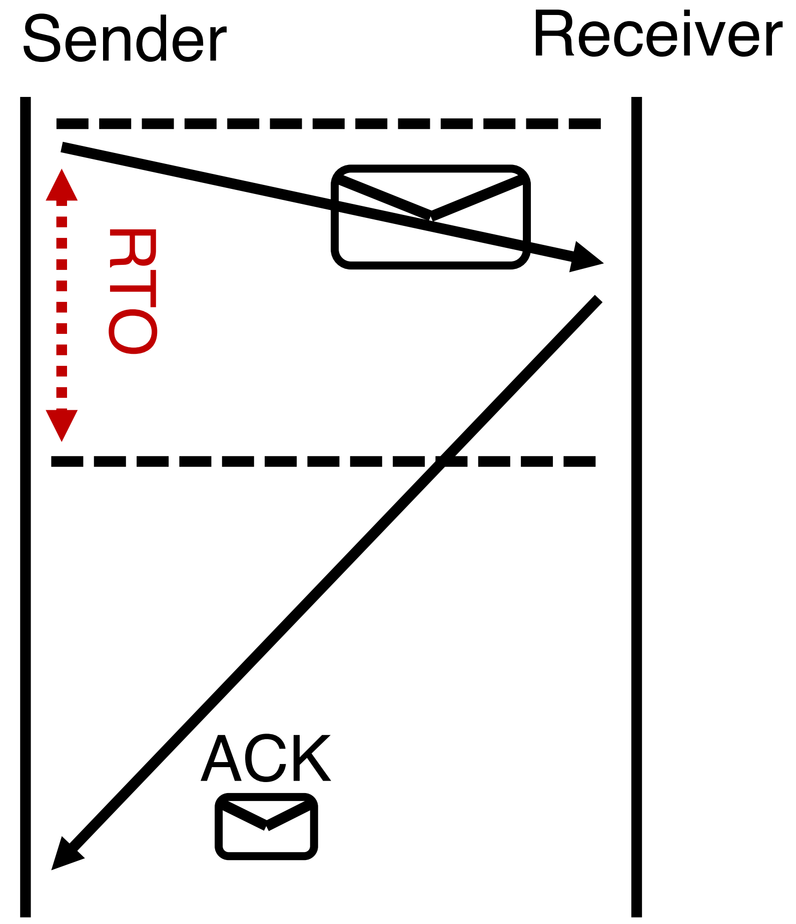
- If sender receives an ACK, it knows that the receiver got the packet.

- What if a packet was lost and ACK never arrives?



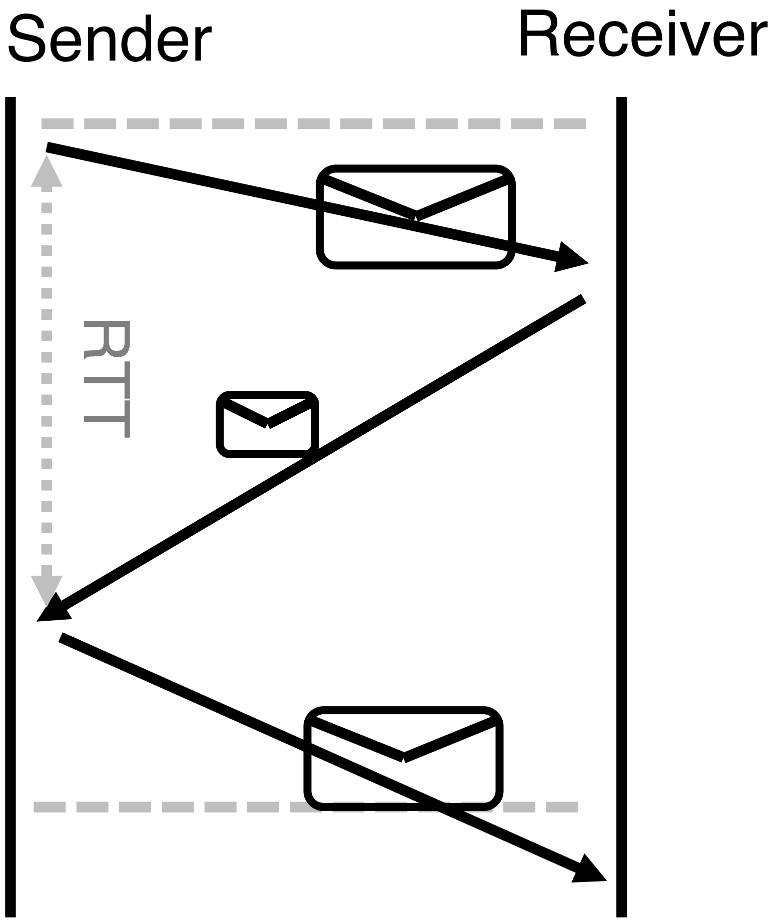
Coping with packet loss: (2) RTO

- Key idea: Wait for a duration of time (called retransmission timeout or RTO) before re-sending the packet
- In TCP, the onus is on the sender to retransmit lost data when ACKs are not received
- Retransmission works also if ACKs are lost or delayed



Sending one packet per ACK enough?

- Should sender wait for an ACK before sending another packet?
- Consider:
  - Round-trip-time: 100 milliseconds
  - Packet size: 12,000 bits
  - Link rate: 12 Mega bits/s
  - Suppose no packets are dropped
- At what rate is the sender getting data across to the receiver?



120 Kilo bit/s (1% of link rate)

Amount of “in-flight” data

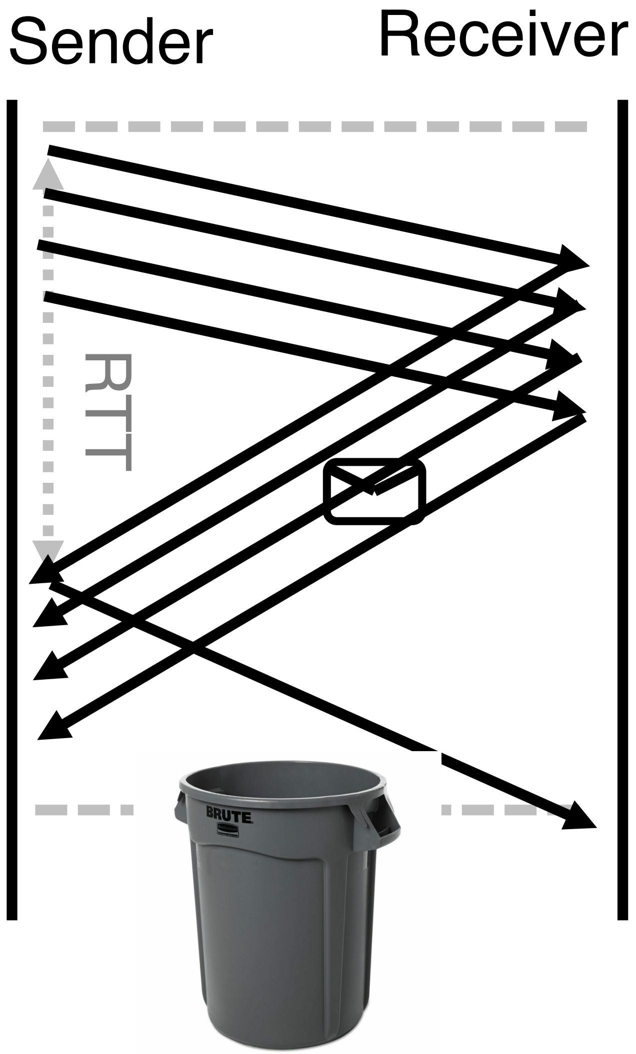
- We term the amount of unACKed data as data “in flight”
- With just one packet in flight, the data rate is limited by the packet delay (RTT) rather than available bandwidth (link rate)
- Idea: Keep many packets in flight!
- More packets in flight improves throughput

Keeping many packets in flight

- In our example before, if there are, say 4 packets in flight, throughput is 480 Kbits/s!

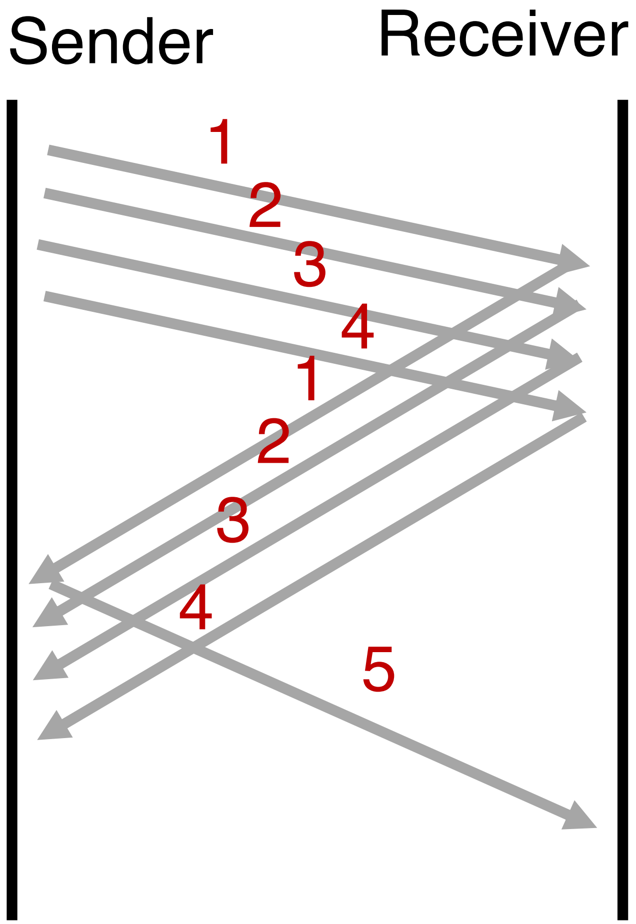
- We just improved the throughput 4 times by keeping 4 packets in flight

- Trouble: what if some packets (or ACKs) are dropped?
- How should the sender retransmit?



Keeping track of packets (and ACKs)

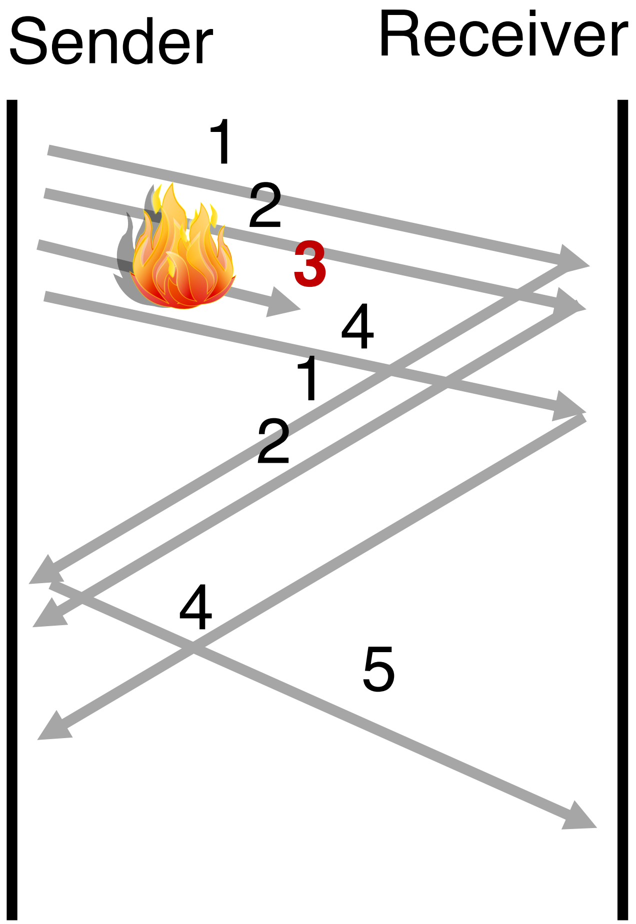
- Every packet contains a sequence number
  - (In reality, every byte has a sequence number)
- ACK echoes the sequence number of the packet that is acknowledged
- Sender retransmits only those packets whose sequence numbers haven't been ACKed



Ordered Delivery

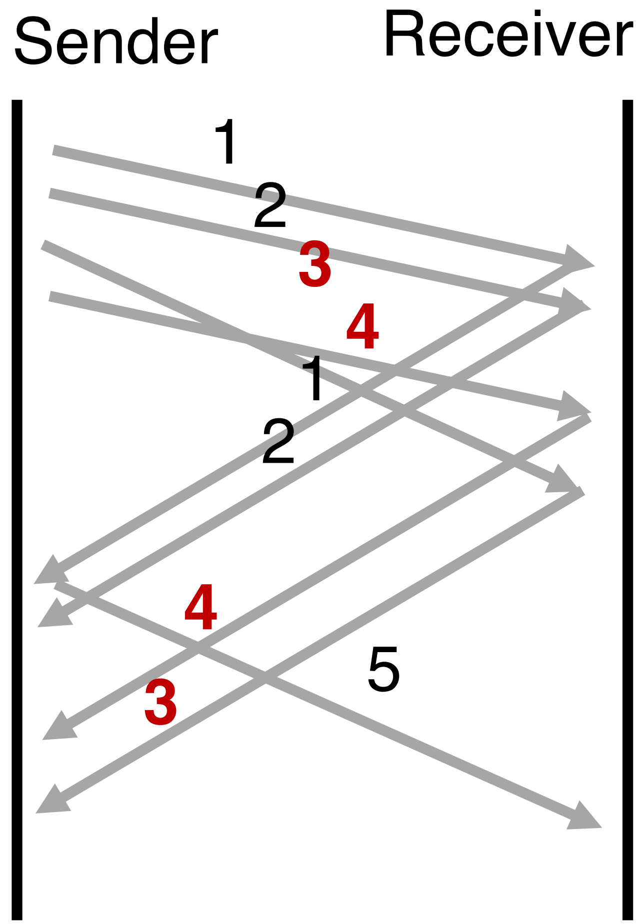
Reordering at the receiver side

- Let's suppose receiver gets packets 1, 2, and 4, but not 3 (dropped)
- Suppose you're trying to download a Word document containing a report
- What would happen if transport at the receiver directly presents packets 1, 2, and 4 to the Word application?

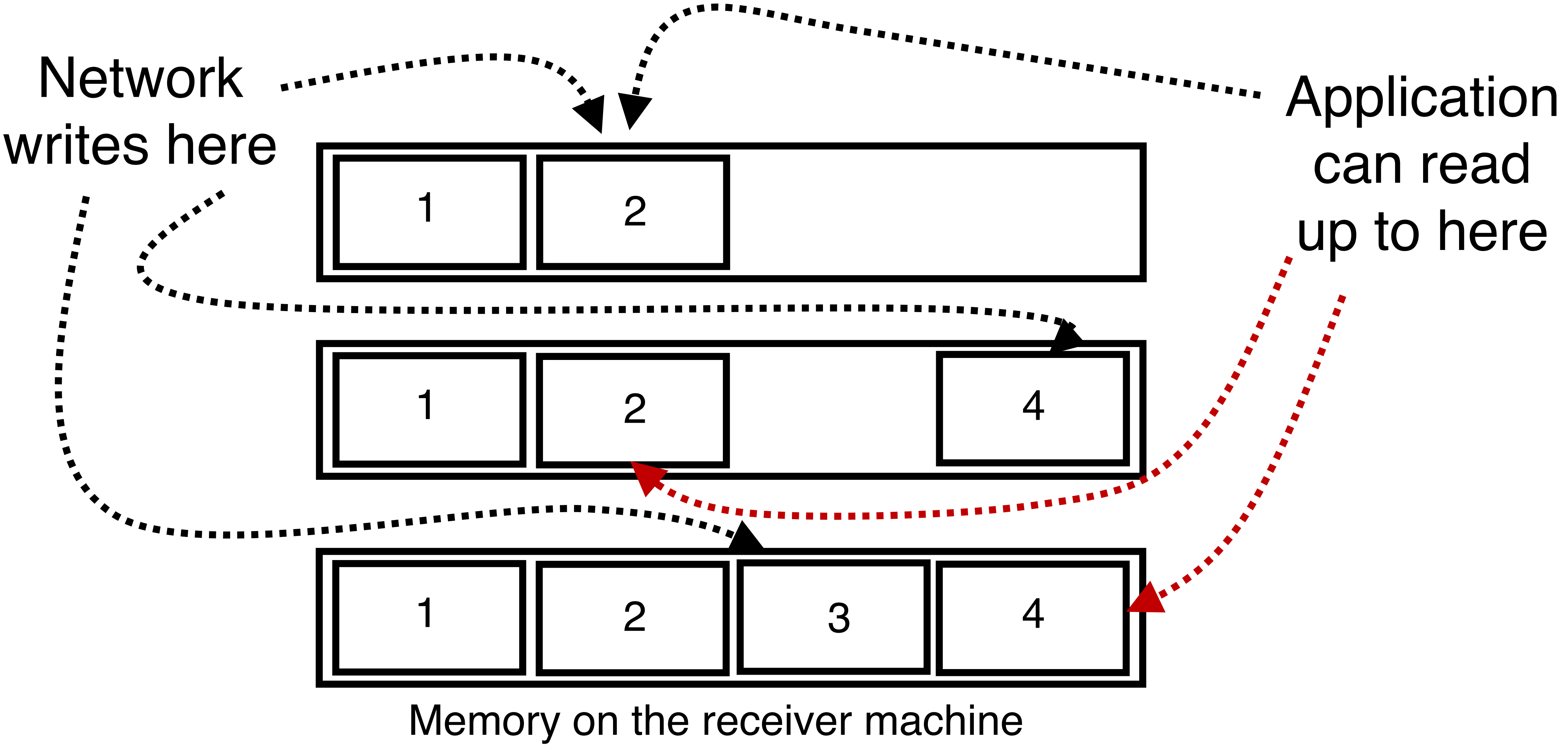


Reordering at the receiver side

- Reordering can also happen due to packets taking different paths through a network
- Receiver needs a general strategy to ensure that data is presented to the application in order of transmission



Buffering at the receiver side



Buffering at the receiver side

- The TCP receiver uses a memory buffer to hold packets until they can be read by the application in order

- This process is known as TCP reassembly

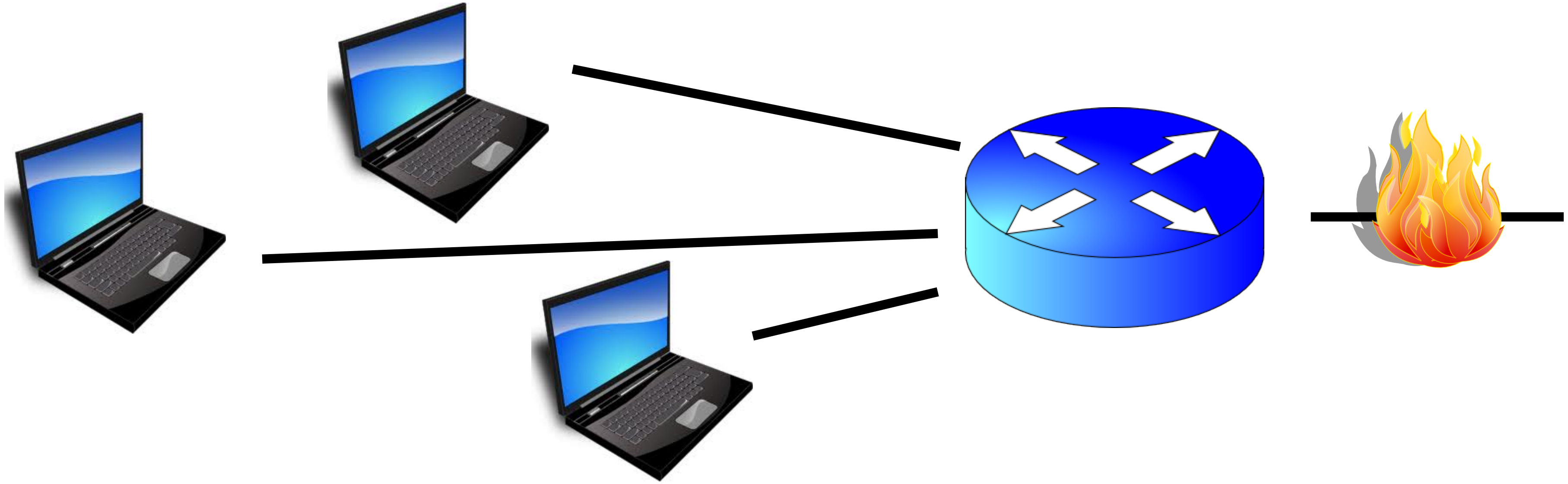
Implications of TCP reassembly

- Packets cannot be delivered to an application if there is an in-order packet missing from the receiver's buffer
  - TCP application throughput will suffer if there is too much packet reordering in the network

- There is only so much the receiver can buffer before dropping subsequent packets (even if successfully arrived at receiver)
  - A TCP sender can only send as much as the free receiver buffer space available before packets are dropped at the receiver
  - This number is called the receiver window size
  - TCP is said to implement flow control

Congestion control

How should multiple endpoints share net?



- It is difficult to know where the bottleneck link is
- It is difficult to know how many other endpoints are using that link
- Endpoints may join and leave at any time
- Network paths may change over time, leading to different bottleneck links (with different link rates) over time

The approach that the Internet takes is to use a distributed algorithm to converge to an efficient and fair outcome.

The approach that the Internet takes is to use a distributed algorithm to converge to an efficient and fair outcome.

No one can centrally view or control all the endpoints and bottlenecks in the Internet.

Every endpoint must try to reach a globally good outcome by itself: i.e., in a distributed fashion.

The approach that the Internet takes is to use a distributed algorithm to converge to an efficient and fair outcome.

If there is spare capacity in the bottleneck link, the endpoint should use it.

The approach that the Internet takes is to use a distributed algorithm to converge to an efficient and fair outcome.

If there are N endpoints sharing a bottleneck link, they should be able to get equitable shares of the link's capacity.

The approach that the Internet takes is to use a distributed algorithm to converge to an efficient and fair outcome.

So, how to achieve this?

Feedback from network offers clues…

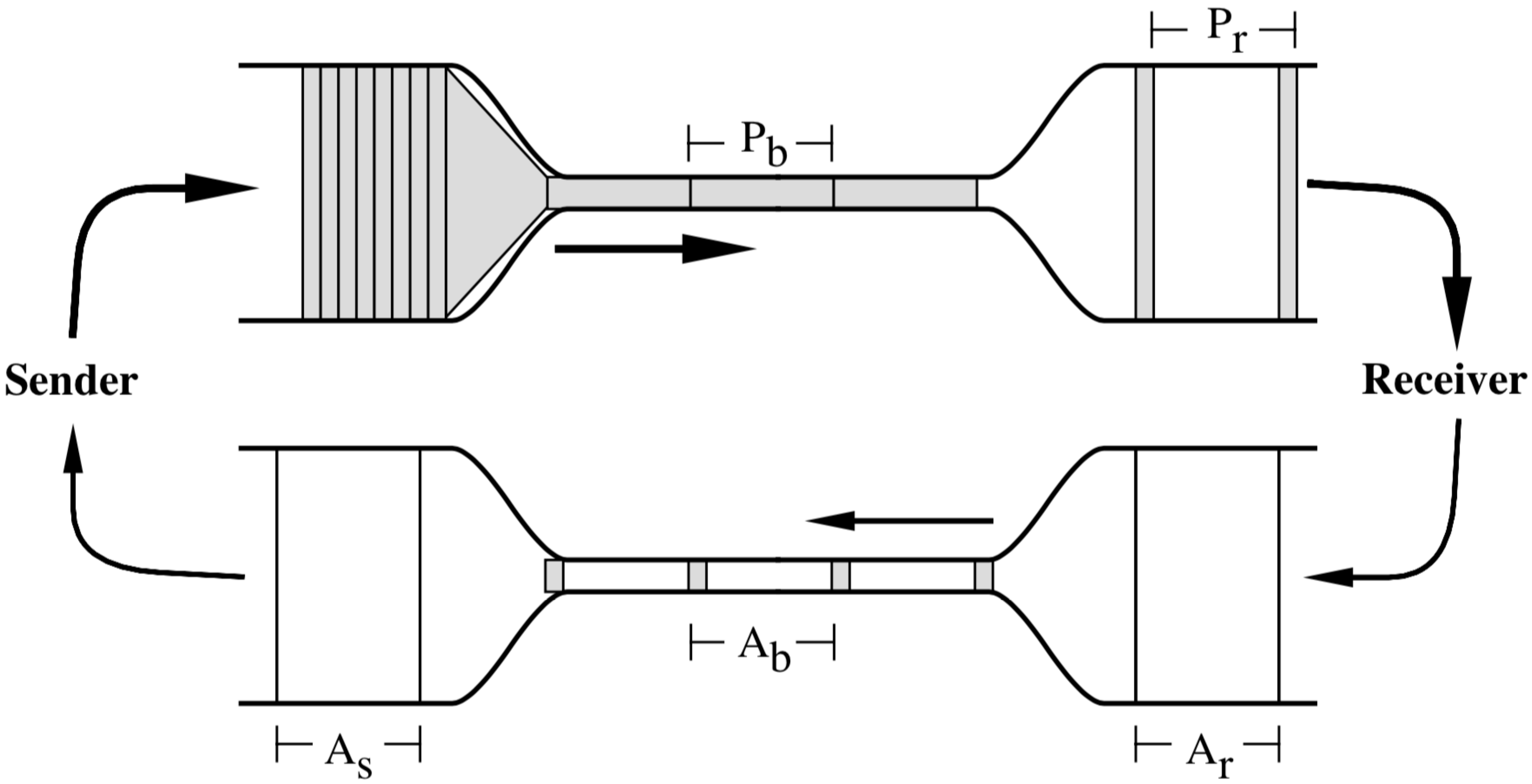
- Signals
  - Packets being dropped (ex, RTO fires)
  - Packets being delayed
  - Rate of incoming ACKs

"Implicit" feedback signals (more on explicit signals later)

- Knobs
  - What can you change to "probe" the sending rate?
  - Suppose receiver buffer is unbounded:
    - The amount of in-flight data is called the congestion window
  - Increase congestion window: e.g., by x or by a factor of x
  - Decrease congestion window: e.g., by x or by a factor of x

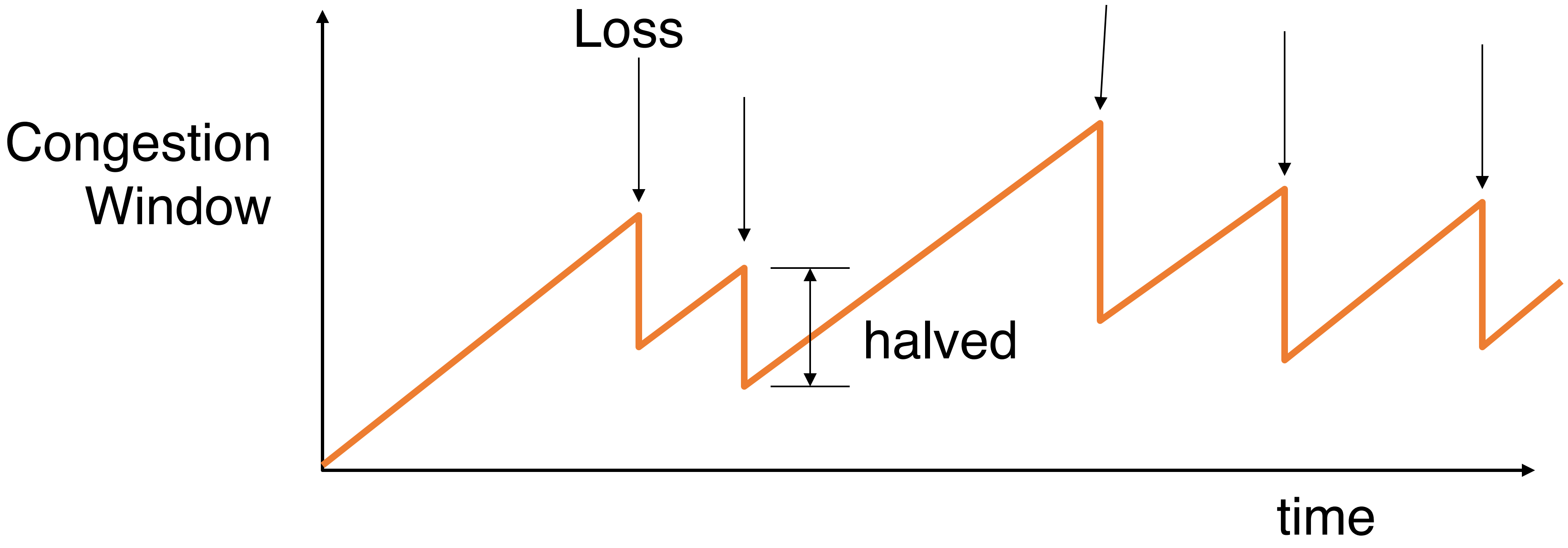
Time for an activity

ACK clocking: steady state behavior



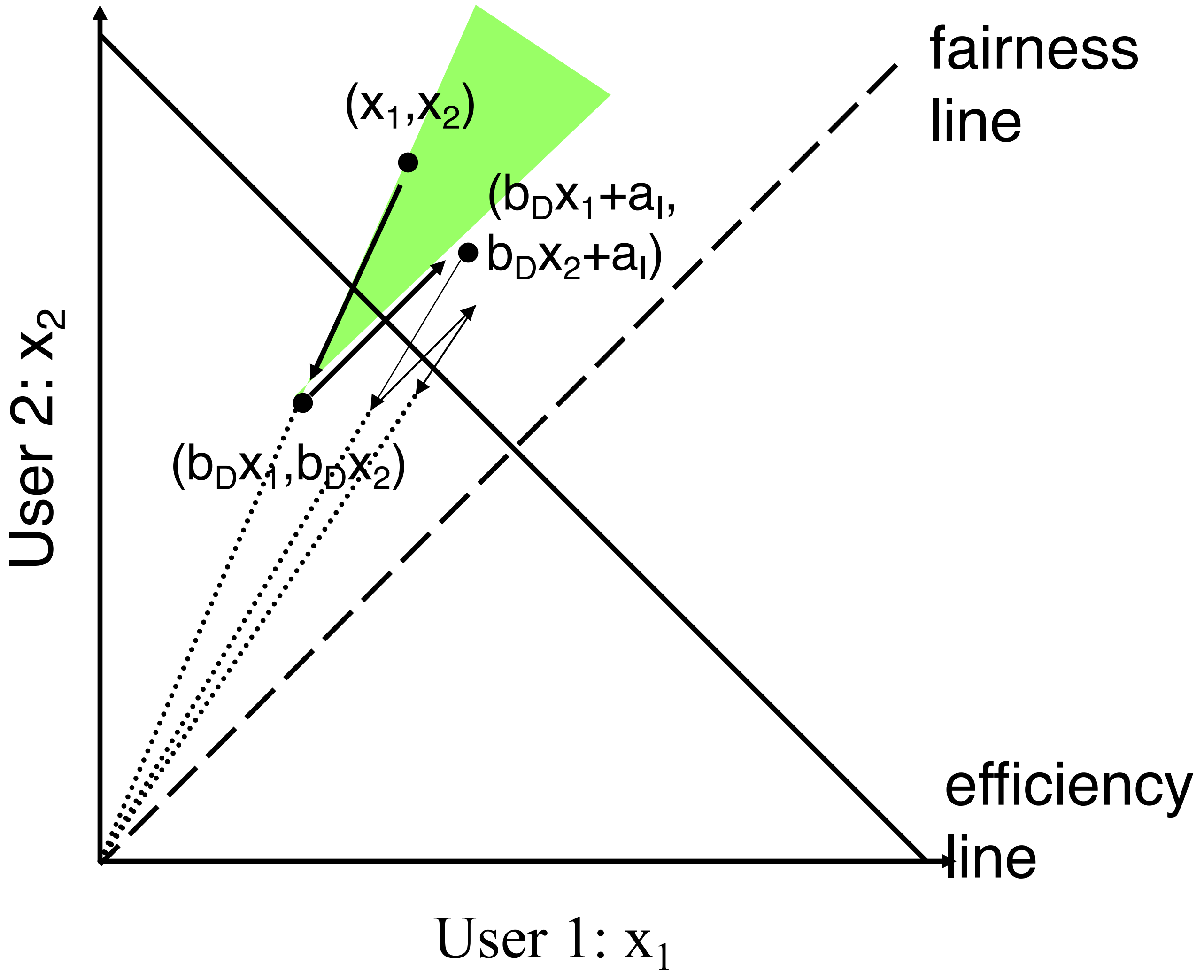
But how to get to steady state?

- Slow start?
- Additive increase, multiplicative decrease (AIMD)



Why AIMD?

- Converges to fairness
- Converges to efficiency
- Increments to rate smaller as fairness increases

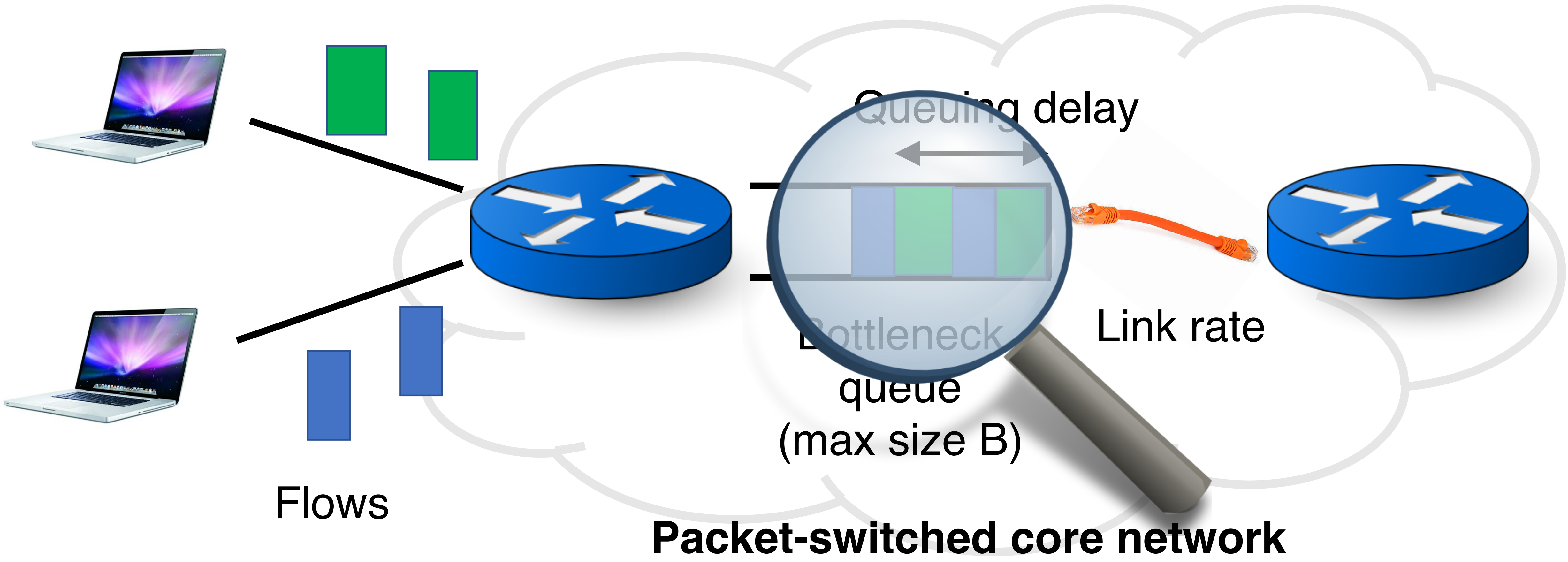


Packet Scheduling

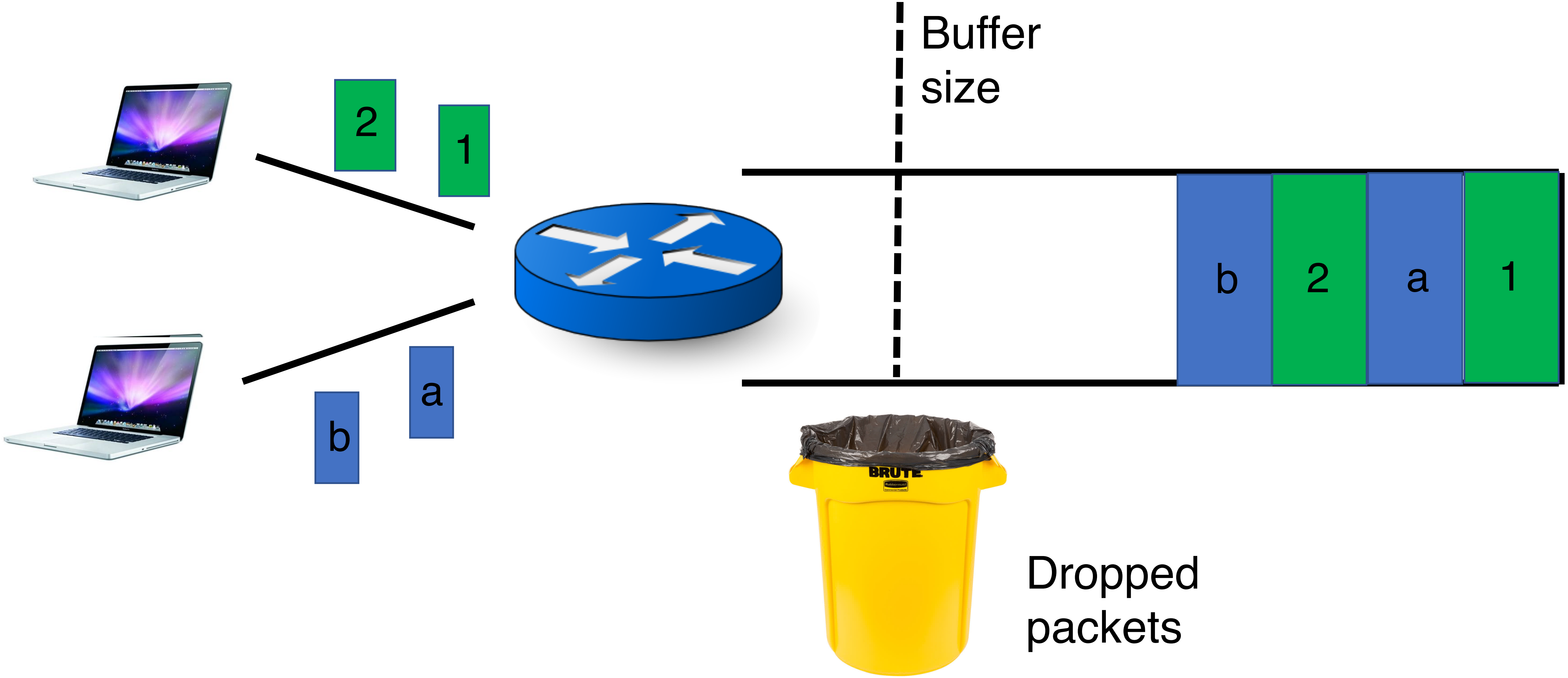
Are endpoint algorithms alone enough?

- What if an endpoint is malicious or buggy?
- Want the network core to do something more about resource allocation than best effort

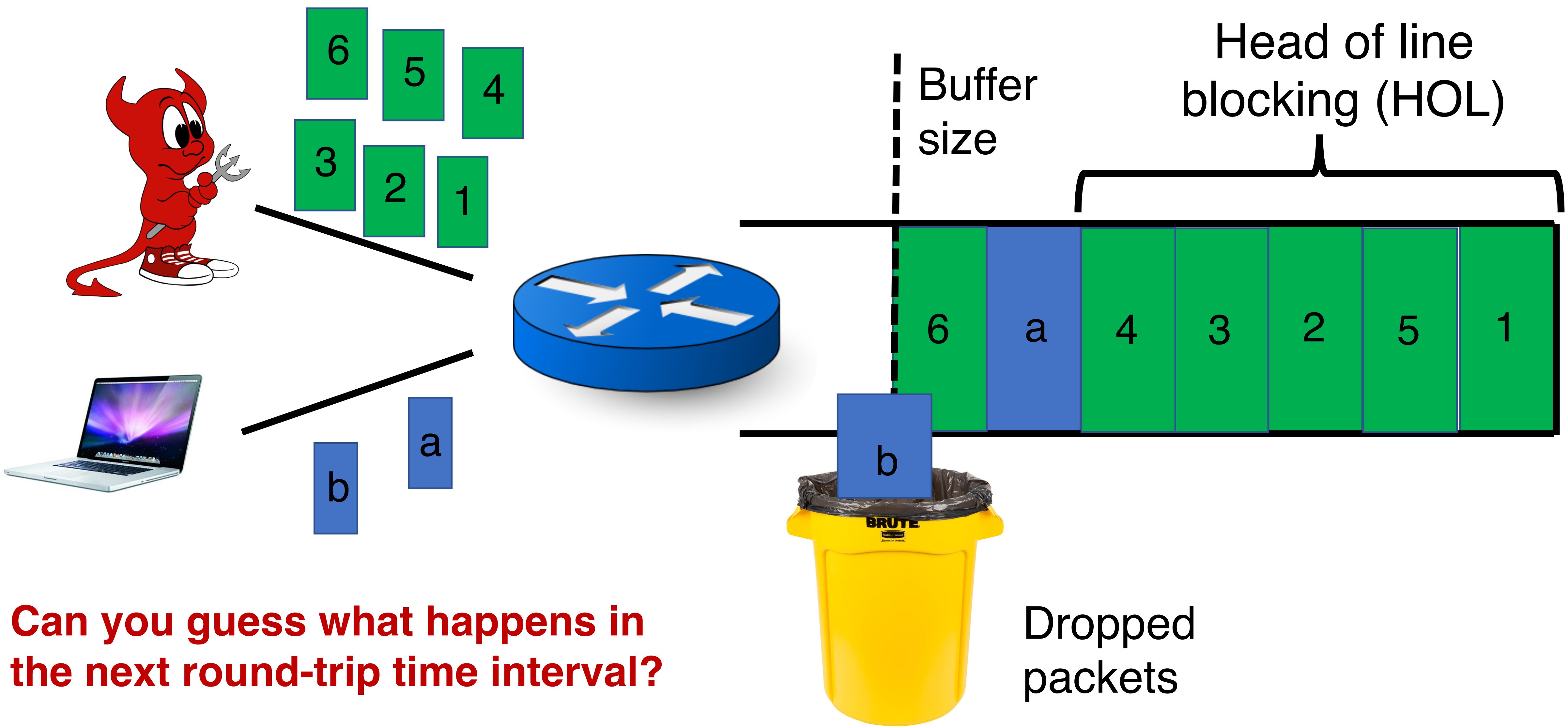
Network model



First-in first-out (FIFO) queue + tail-drop

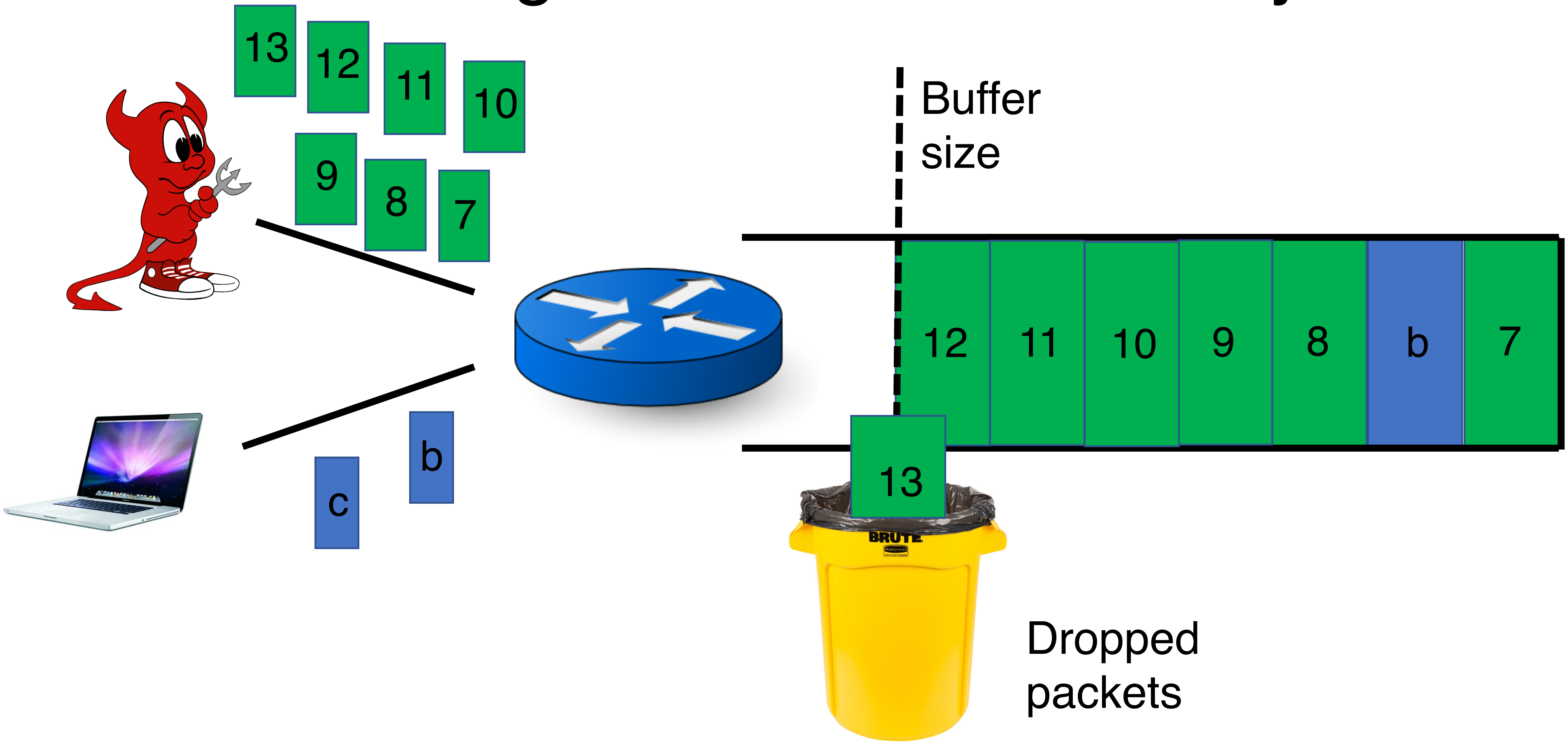


First-in first-out (FIFO) queue + tail-drop

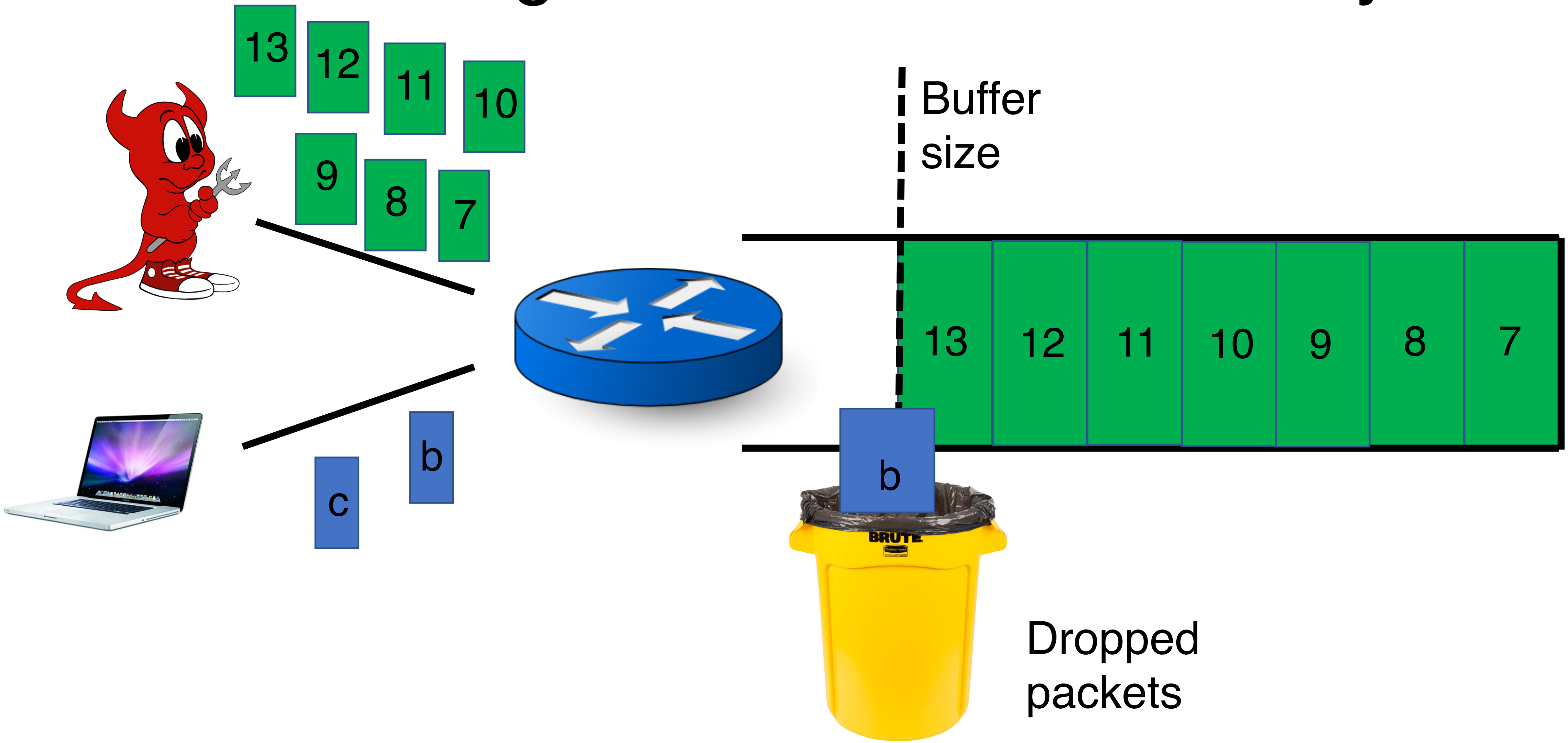


Can you guess what happens in the next round-trip time interval?

ACK-clocking makes it worse: lucky case



ACK-clocking makes it worse: unlucky case



Network monopolized by “bad” endpoints

- An ACK signals the source of a free router buffer slot
  - Further, ACK clocking means that the source transmits again
- Contending packet arrivals may not be random enough
  - Blue flow can’t capture buffer space for *a few* round-trips
- Sources which sent successfully earlier get to send again
- A FIFO tail-drop queue *incentivizes* sources to misbehave!

Packet scheduling on routers

- We will discuss packet scheduling algorithms implemented on routers in detail later in this course.

- Goal: Achieve a predetermined resource allocation regardless of endpoint behavior

- How to make such allocation "efficient"?
  - Implement on routers at high speeds
  - Achieve equitable sharing of network bandwidth & queues
  - Use available bandwidth effectively