Administrivia

- Review 1 due tomorrow
    - Email your reviews to me

- Office hours on Thursdays 10—12

- MUD: Send me your top 1—3 questions on this lecture

- Guest lectures next week by Prof. Richard Martin
    - Class slides by 10 mins (starts at 8.50 am) on both days

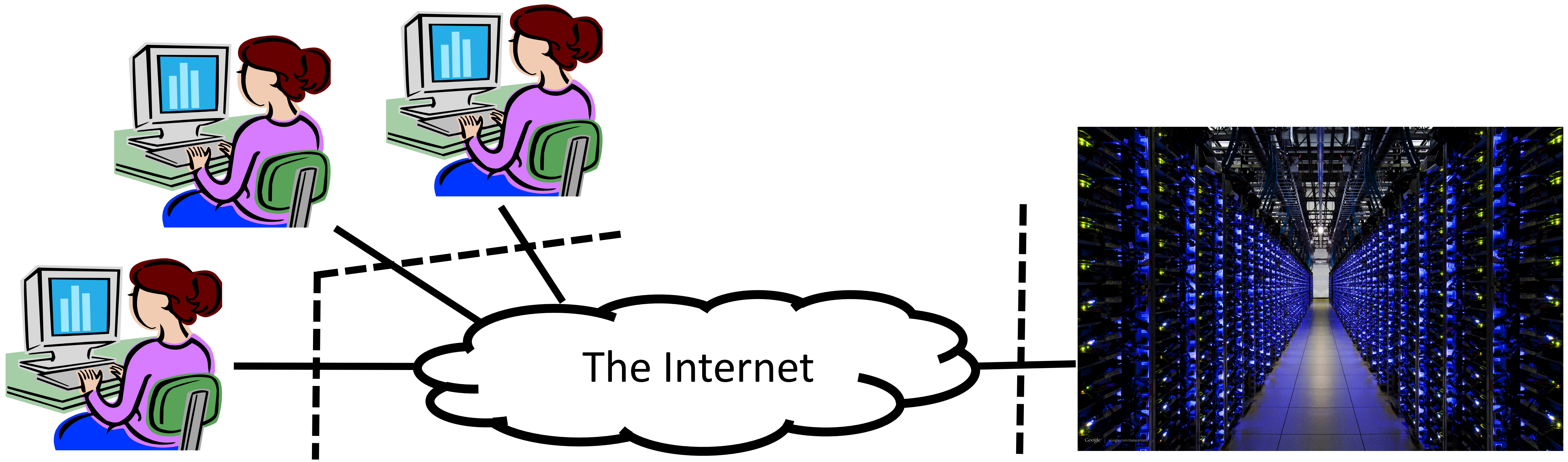
Packet Scheduling

Lecture 5, Computer Networks (198:552)

RUTGERS
UNIVERSITY | NEW BRUNSWICK

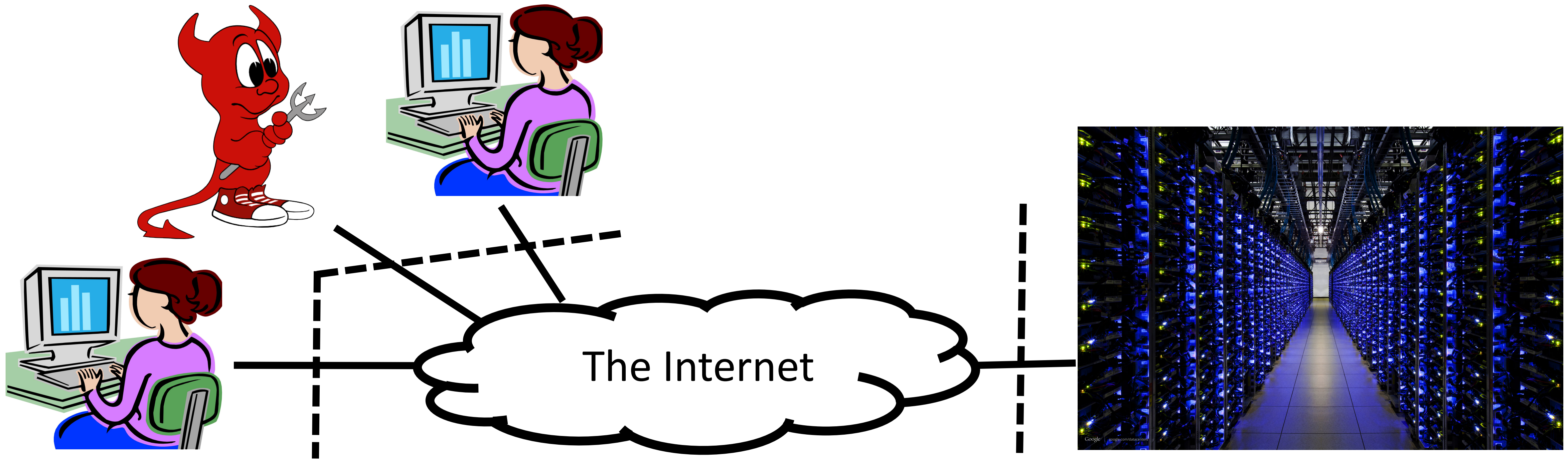
Resource allocation in the core

- Edge: Run transports to provide app guarantees
- Core: Transmit packets with best-effort guarantees

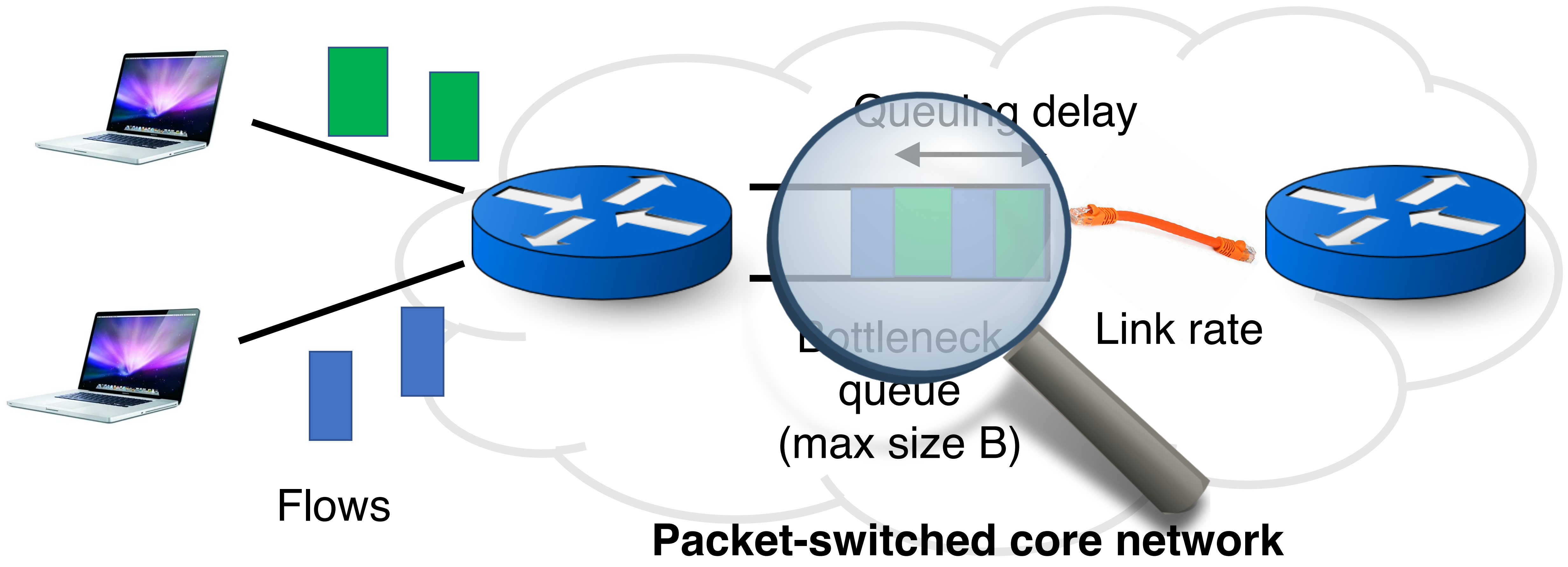


Resource allocation in the core

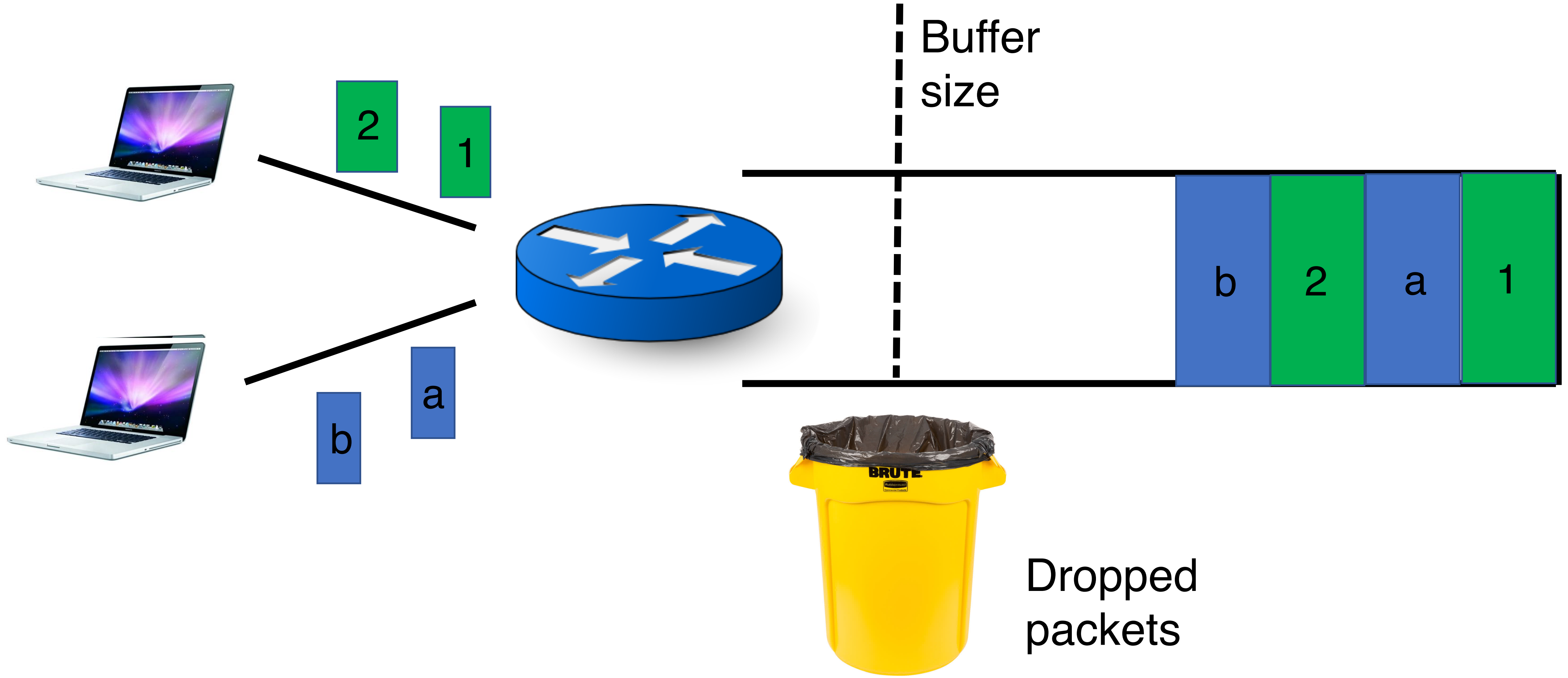
- Edge: Run transports to provide app guarantees
- Core: Transmit packets with best-effort guarantees
- But what about malicious or buggy endpoints?



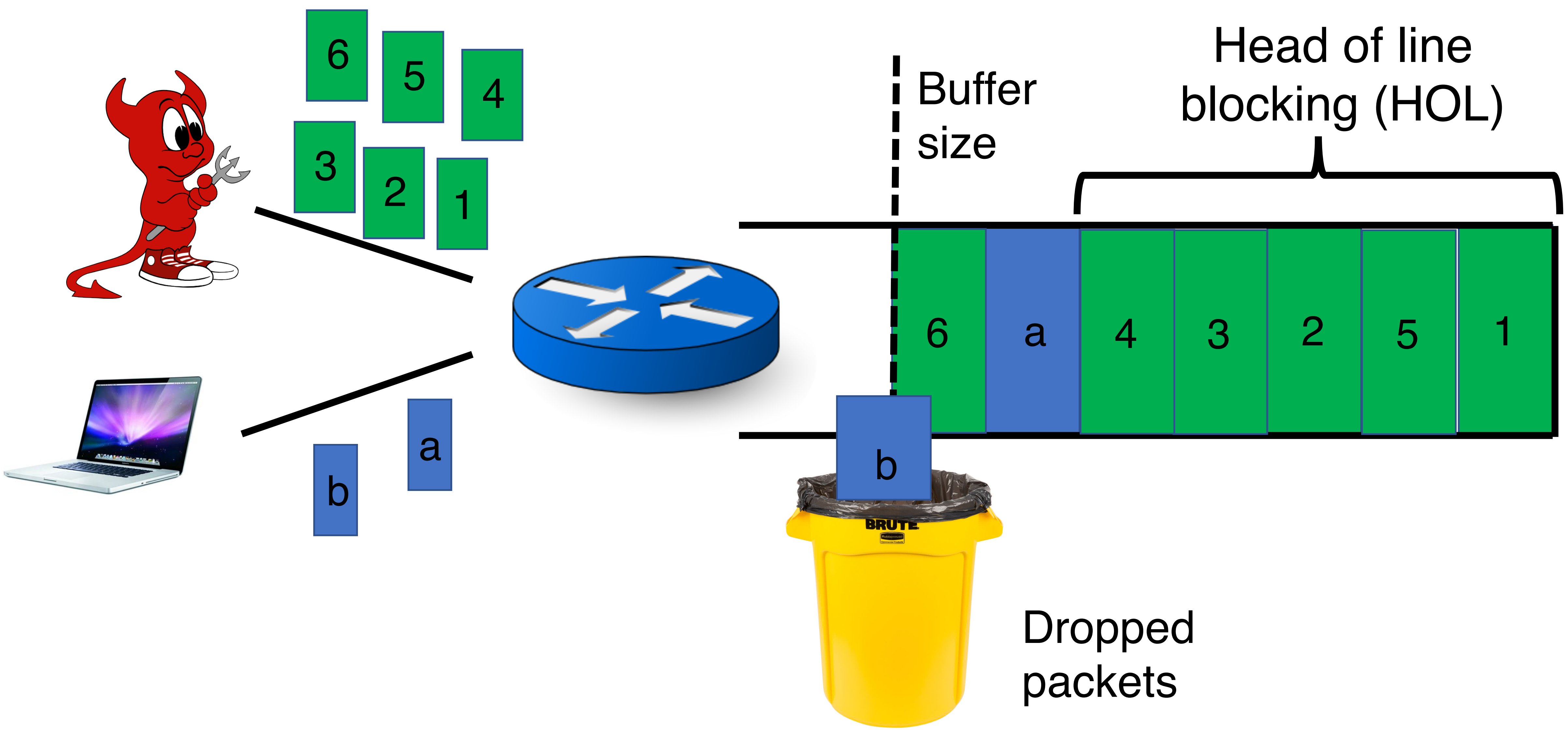
Network model



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

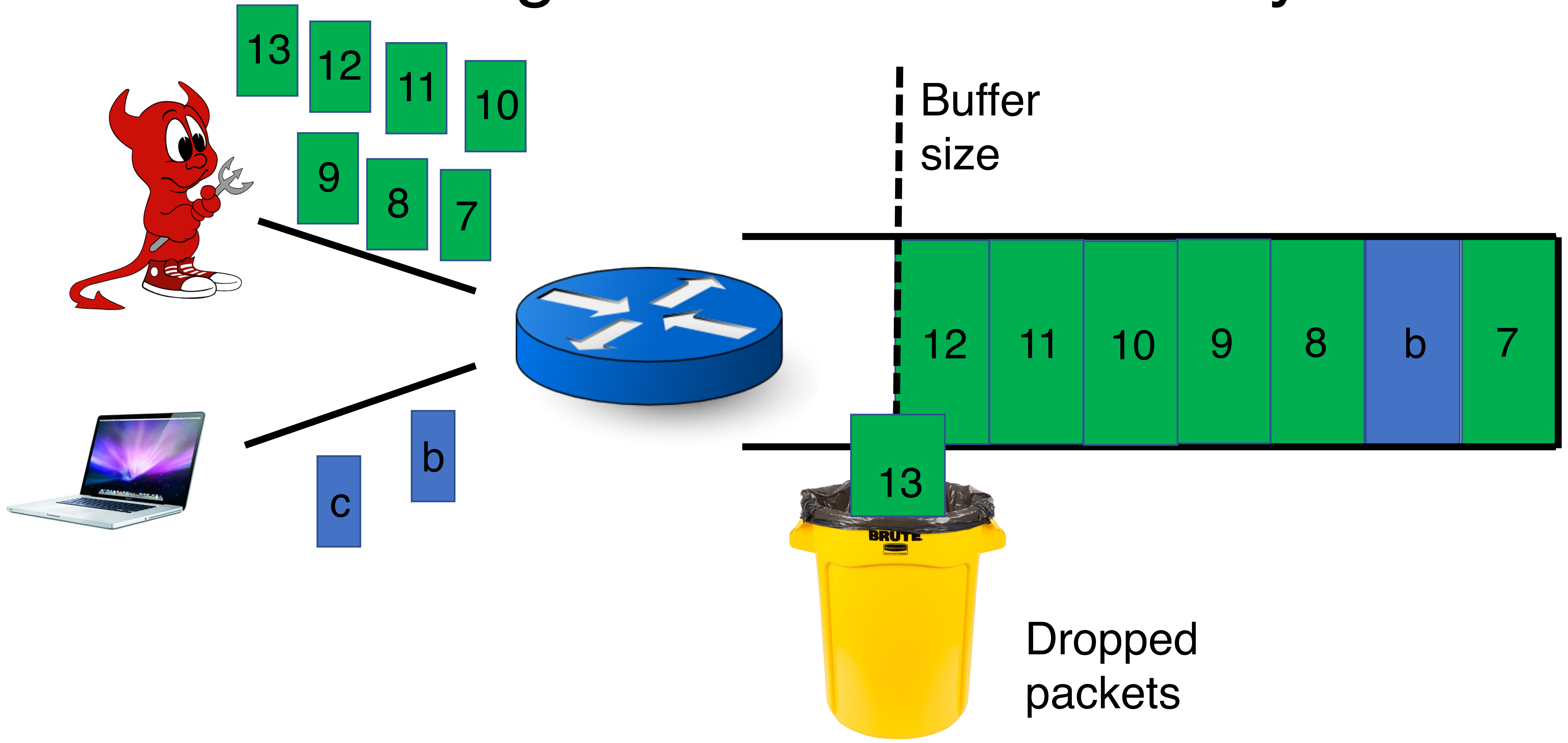


First-in first-out (FIFO) queue + tail-drop

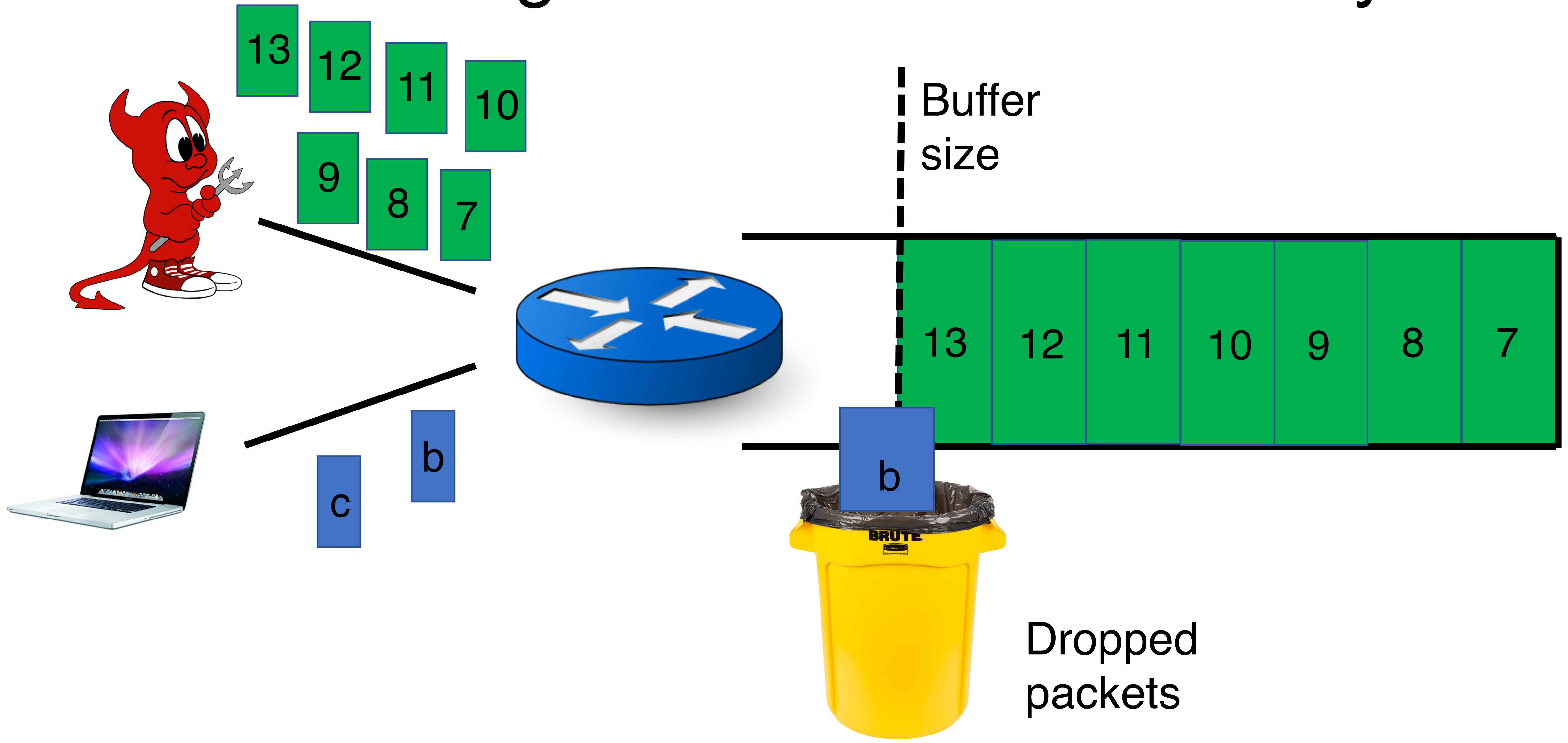


What happens in the next round-trip time interval?

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!

Goal: Better resource sharing in the core

- What’s a fair resource allocation?

- How to achieve a predetermined resource allocation?
  - … regardless of source behavior?

- How to make the allocation “efficient”?
  - Use the available bandwidth effectively
  - Build routers that work at high link rates
  - Maybe even be a little unfair to apps to be more efficient overall

Fair Resource Allocation

Allocate *how?* among *who*?

Fair and efficient use of a resource

- Suppose *n* users share a single resource
    - Like the bandwidth on a single link
    - E.g., 3 users sharing a 30 Gbit/s link

- What is a *fair* allocation of bandwidth?
    - Suppose user demand is "elastic" (i.e., unlimited)
    - Allocate each a *1/n* share (e.g., 10 Gbit/s each)

- But, "equality" is not enough
    - Which allocation is best: [5, 5, 5] or [18, 6, 6]?
    - [5, 5, 5] is more "fair", but [18, 6, 6] more efficient
    - What about [5, 5, 5] vs. [22, 4, 4]?

Fair use of a single resource

- What if some users have *inelastic* demand?
  - E.g., 3 users where 1 user only wants 6 Gbit/s
  - And the total link capacity is 30 Gbit/s
- Should we still do an “equal” allocation?
  - E.g., [6, 6, 6]
  - But that leaves 12 Gbps unused
- Should we allocate in proportion to demand?
  - E.g., 1 user wants 6 Gbps, and 2 each want 20 Gbit/s
  - Allocate [4, 13, 13]?
- Or, give the least demanding user all she wants?
  - E.g., allocate [6, 12, 12]?

Max-min fairness

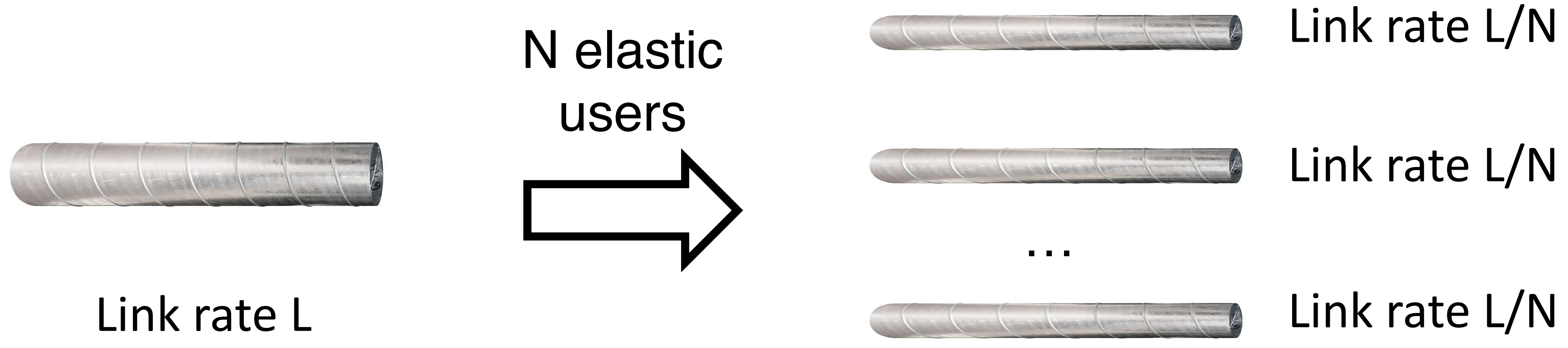
- Protect the less fortunate
    - Any attempt to *increase* the allocation of one user
    - … necessarily *decreases* the allocation of another user with equal or lower allocation

- Fully utilize a “bottlenecked” resource
    - If demand exceeds capacity, the link is fully used

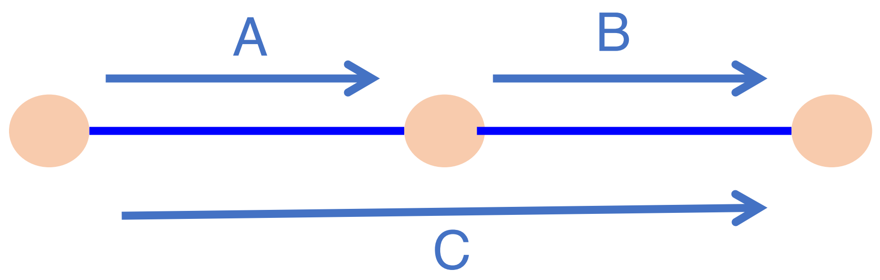
Max-min fairness

- Progressive filling algorithm
  - Grow all rates until some users stop having demand
  - Continue increasing all remaining rates until link is fully utilized

- If all users have elastic demands, single resource shared evenly



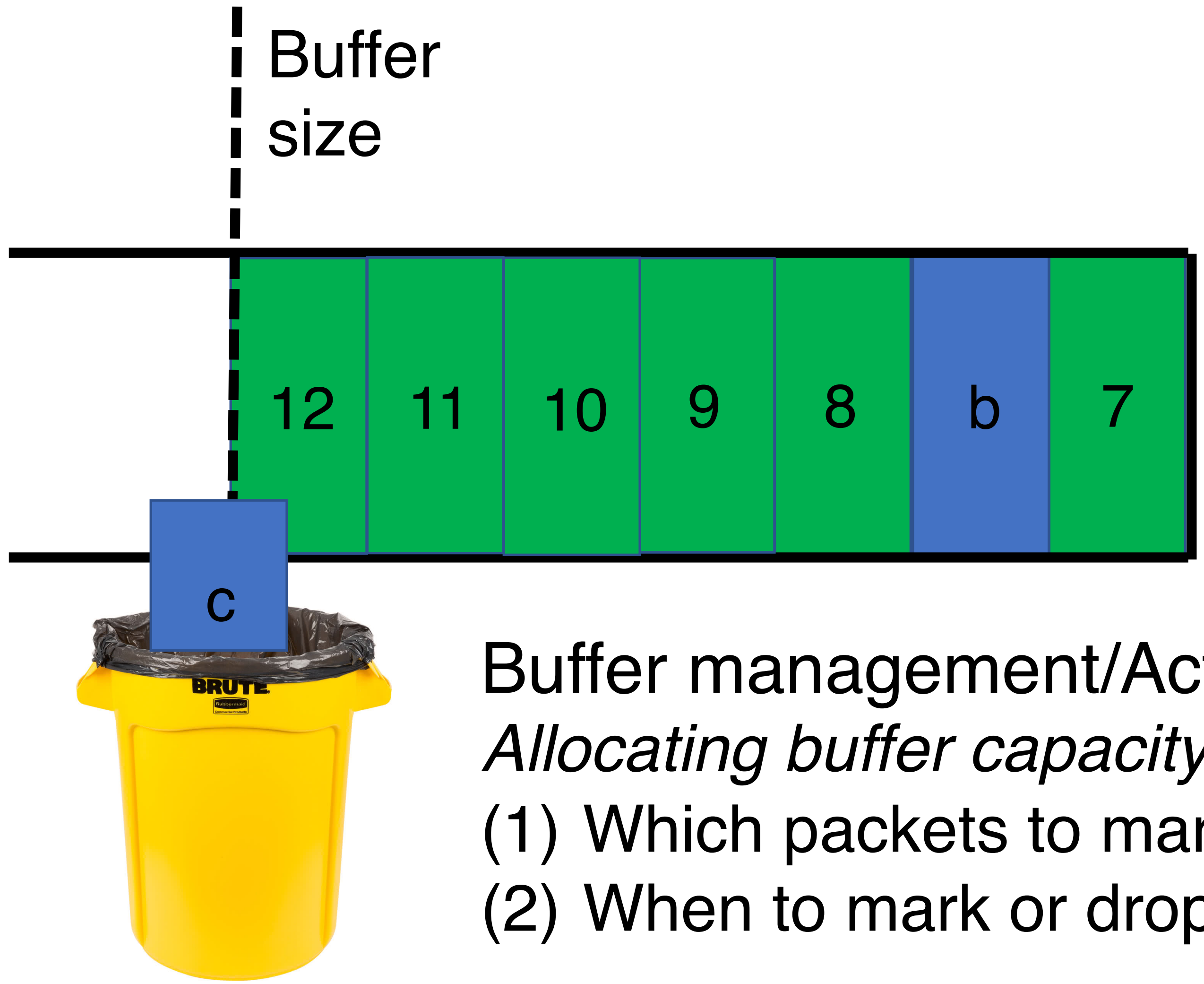
Resource Allocation Over Paths



Three users A, B, and C
Two 30 Gbit/s links

- Maximum throughput: [30, 30, 0]
  - Total throughput of 60, but user C starves
- Max-min fairness: [15, 15, 15]
  - Equal allocation, but throughput of just 45
- *Proportional fairness*: [20, 20, 10]
  - Balance trade-off between throughput and equality
  - Throughput of 50, and penalize C for using 2 busy links

Scheduling and Buffer Management



Scheduling:
Allocating link bandwidth & queueing delays
(1) Which packet to send next?
(2) When to send the next packet?

Buffer management/Active Queue Management (AQM)
Allocating buffer capacity
(1) Which packets to mark or drop?
(2) When to mark or drop?

Allocate fairly among *who?*

Abstract entity:
a *flow*

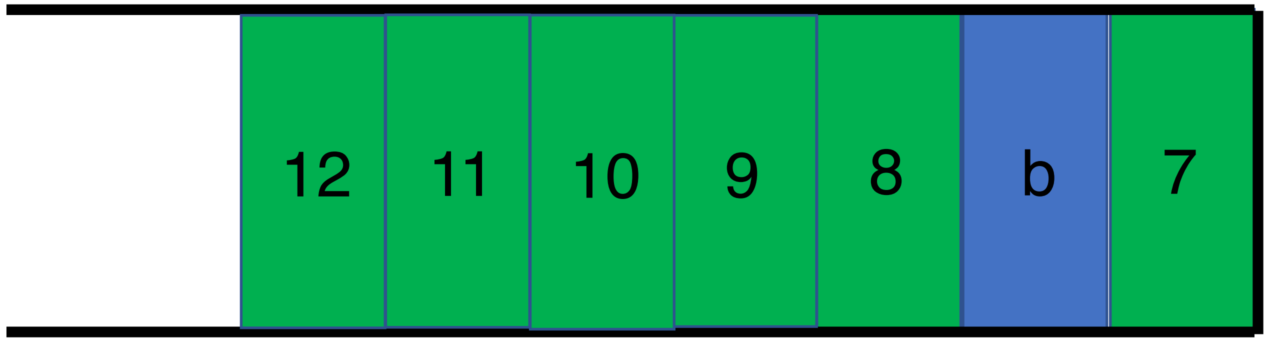
- Traffic sources?
  - Web servers, video servers, etc. need more than their fair share
- Traffic destinations?
  - Vulnerable to malicious sources denying service to receivers
- Source-destination pairs?
  - Can open up connections to many destinations
- Application flows? (i.e., src + dst + transport ports)
  - Malicious app can open up many such flows
- Administrative entities? (e.g., Rutgers NetID, ISP, …)
  - How is a router to identify packets belonging to an entity?

Scheduling Algorithms

Which packet to send next?

When to send the next packet?

A taxonomy



- Granularity of allocation
  - Per-packet vs. per-flow vs bit-by-bit
- Pre-emptive vs. non-pre-emptive
  - Do you interrupt the current packet/flow if another shows up?
- Size-aware vs. unaware
  - Do you take into account flow or packet sizes in scheduling?
- Class-based vs. shared
  - Are some flows strictly higher priority than others?
- Work-conserving vs. non-conserving
  - Do you always use spare link capacity when there is demand?
- How complex is the implementation?

Common scheduling algorithms (1/2)

- FIFO over packets (example in previous slides)
- Round-robin over packets of different flows
  - You would have seen these in the FQ & DRR papers
- Shortest Remaining Processing Time (SRPT)
  - Flow-size-aware allocation which strictly prioritizes short flows
  - Flow-size-unaware variant may “predict” demand using a known flow size distribution

Common scheduling algorithms (2/2)

- Processor sharing
  - Assume each flow gets a fair share of the link every unit of time
  - Ideal: each flow starts receiving service *immediately upon arrival*
- Rate limiting
  - Non-work-conserving: flow can’t send even if more demand than limit
- Class-based prioritization
  - Pre-determined flow classes with strict priorities over each other
  - Starve low priority flows if higher priority flows are always sending
- There are many variants and combinations of these!

Exercise: When does a flow finish?

- Consider a workload mix of “long” and “short” flows arriving at a Q
  - Ex: A flow may have as few as 2 packets or as many as 10^5
- Suppose a scheduling algorithm provides each flow:
  - An average per-packet delay d (e.g., 50 ms)
  - An average link bandwidth share t (e.g., 10 Mbit/s)
- Which among d & t determines when a short flow finishes? Why?
  - How about a long flow?
- If you don’t know the workload mix, is there an optimal sch. alg.?
  - If so, which one? If not, why not?

Analysis and Simulation of a Fair Queueing Algorithm

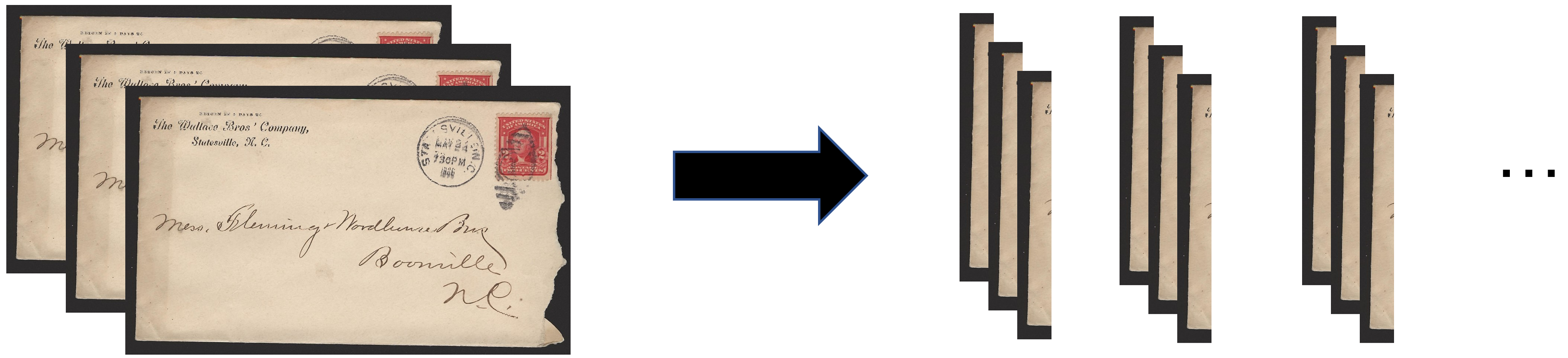
ACM SIGCOMM ’89

Alan Demers, Srinivasan Keshav, and Scott Shenker

An ideal to emulate: Processor sharing

- Fair-share bandwidth in the most fine-grained fashion possible
  - If there are N active flows, each flow gets 1/Nth of the link rate
  - “Bit by bit round robin” (BR)

- Implementing BR directly on routers is unrealistic. Why?
  - One reason: consider the processing of the bit downstream
  - E.g., where to route the bit?



Emulate bit-by-bit round robin (BR)?

- How about round robin over packets?

- Unfair! A flow can use larger packets and gain larger bandwidth

- Instead, determine when a packet would finish with BR
  - Depends only on packet arrival time & # of active flows
  - Let’s call this the “virtual finish time”

- FQ: Transmit packets in the order of the virtual finish times
  - Buffer management: drop pkt of the flow with the largest backlog

How close is FQ to BR?

- Total # bytes transmitted by a flow in BR and FQ aren't different by more than the maximum packet size
    - Independent of the number of active flows!
- Why?

- A "slack" parameter, delta, used to reduce the virtual finish time for inactive flows
    - Doesn't affect long-term throughput

Discussion of FQ

- Is the slack parameter the best method to handle “delay allocation”?
  - What are the pros and cons of this method?
  - Can you think of other methods?
- If there are N active flows, how much per-packet work does FQ need to do to determine the next packet to schedule?
- Why does the performance of a scheduling algorithm depend on the endpoint algorithm?
- What happens in a network of FQs?
  - Is there a variant that considers path-level resource usage?

Efficient Fair Queueing using Deficit Round Robin

ACM SIGCOMM ’95

M. Shreedhar and George Varghese

Problem: Complexity of implementing FQ

- If there are N active flows, FQ needs to do O(log N) work to fetch the next packet to transmit
- Reason: must maintain a sorted list of virtual finish times of *at least the first packet* of each active flow
- Can we avoid maintaining a sorted list somehow?

Idea: Keep track of pkt size unfairness

- Basic idea: round robin over packets of different flows
- Maintain an active list of flows; hash flows to a queue
- Maintain a “quantum” for each queue that is refilled every round
    - By the maximum packet size
- Transmit as many packets as allowed by the queue’s quantum
- If a packet is larger than the quantum, the flow keeps the quantum

Discussion of DRR

- Assume N active flows
- Why is the quantum added every round the maximum pkt size?
- What are the throughput guarantees provided by DRR?
    - Short-term guarantees? Asymptotic?
    - Elastic demands? Inelastic demands?
- What are the delay guarantees provided by DRR?
    - Worst case: may wait until N maximum size packets transmitted!
    - Combine with class-based (strict) prioritization + contracts
- How would you maintain an active flow list in hardware?

Administrivia

- Review 1 due tomorrow
  - Email your reviews to me
- Office hours on Thursdays 10—12
- MUD: Send me your top 1—3 questions on this lecture
- Guest lectures next week by Prof. Richard Martin
  - Class slides by 10 mins (starts at 8.50 am) on both days