Queue Management

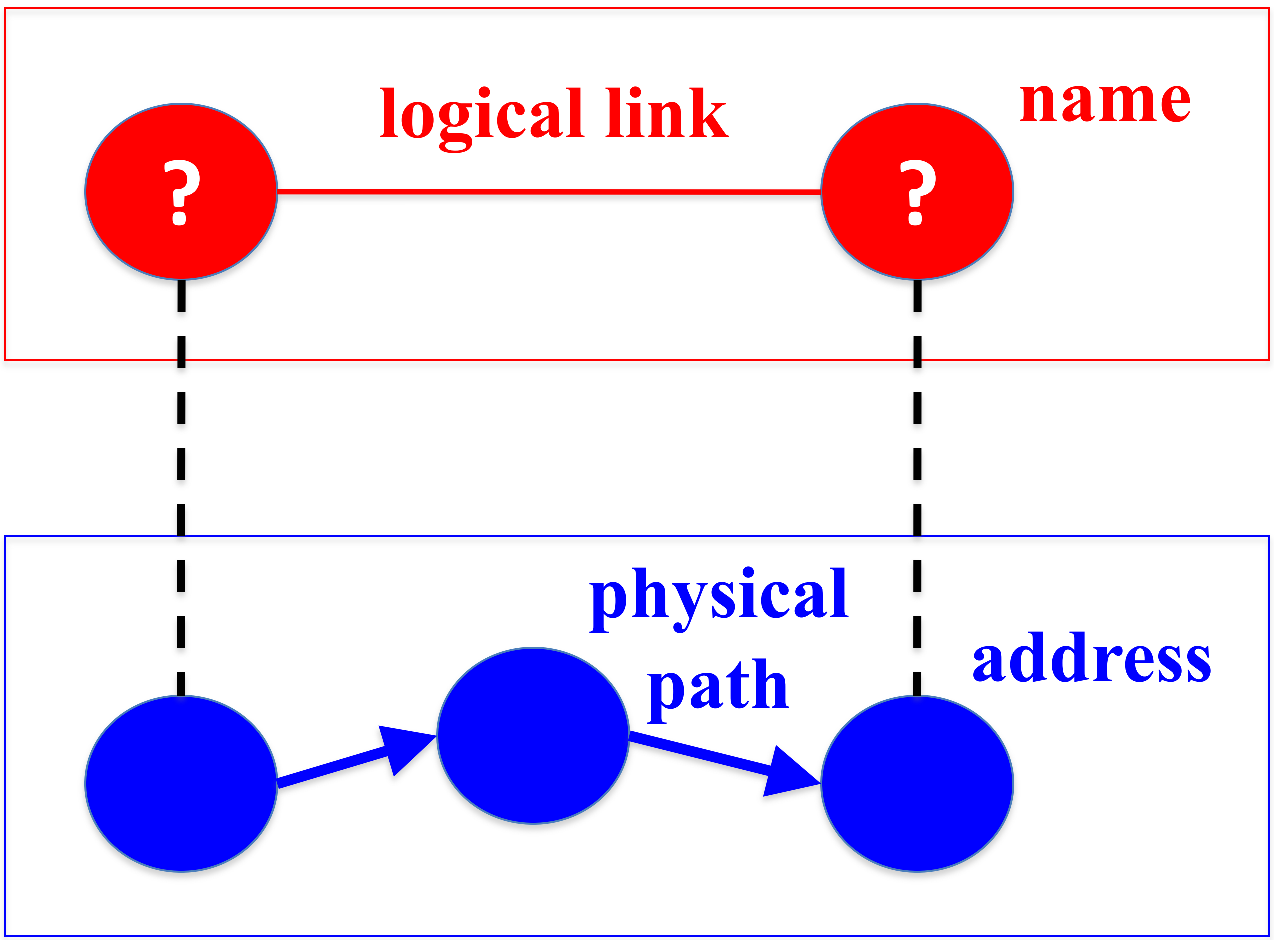
Kyle Jamieson

COS 461: *Computer Networks*

www.cs.princeton.edu/courses/archive/fall20/cos461/

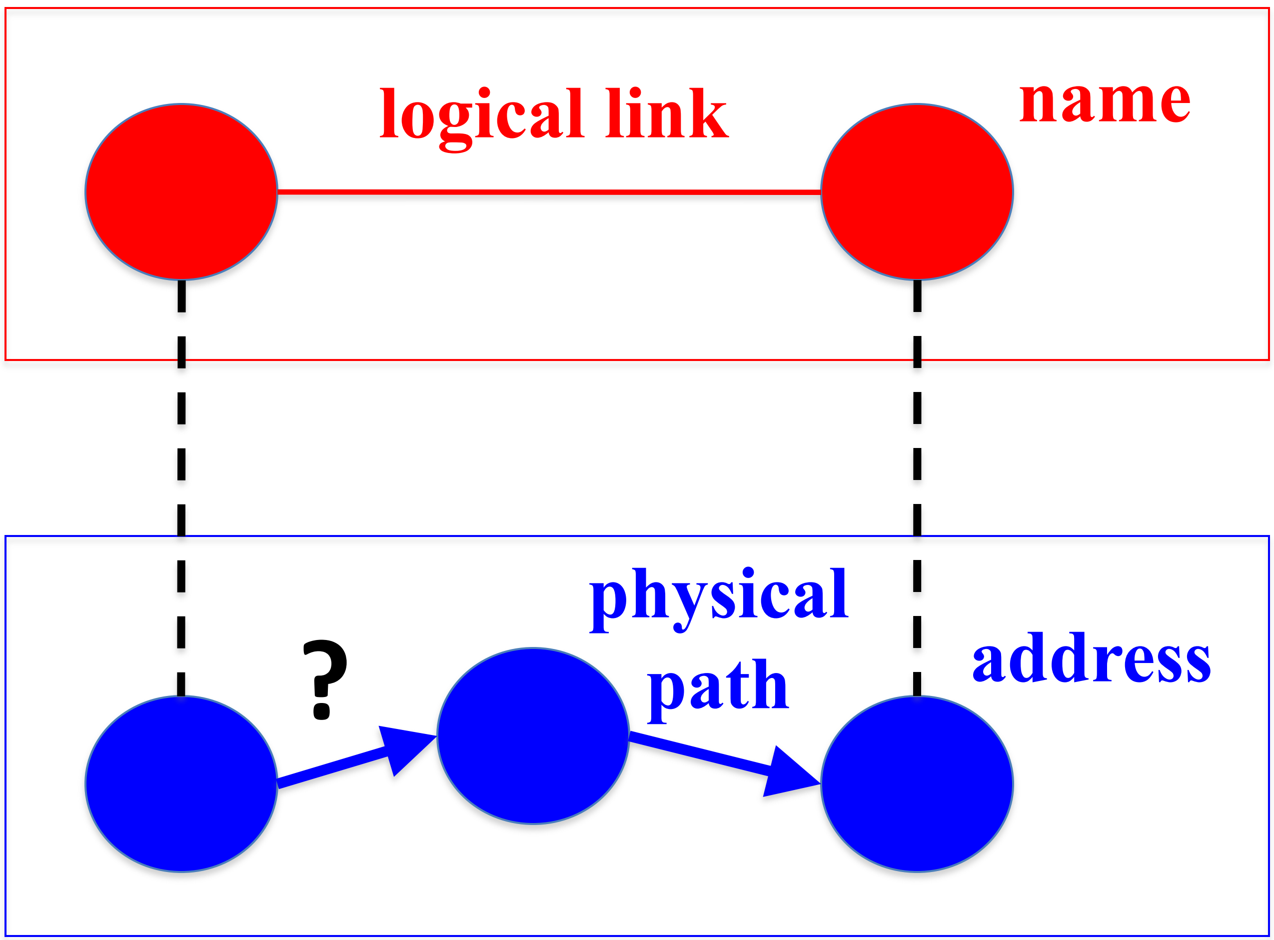
Last lecture: Congestion Control

What can *end-points* do to collectively to make good use of shared underlying resources?



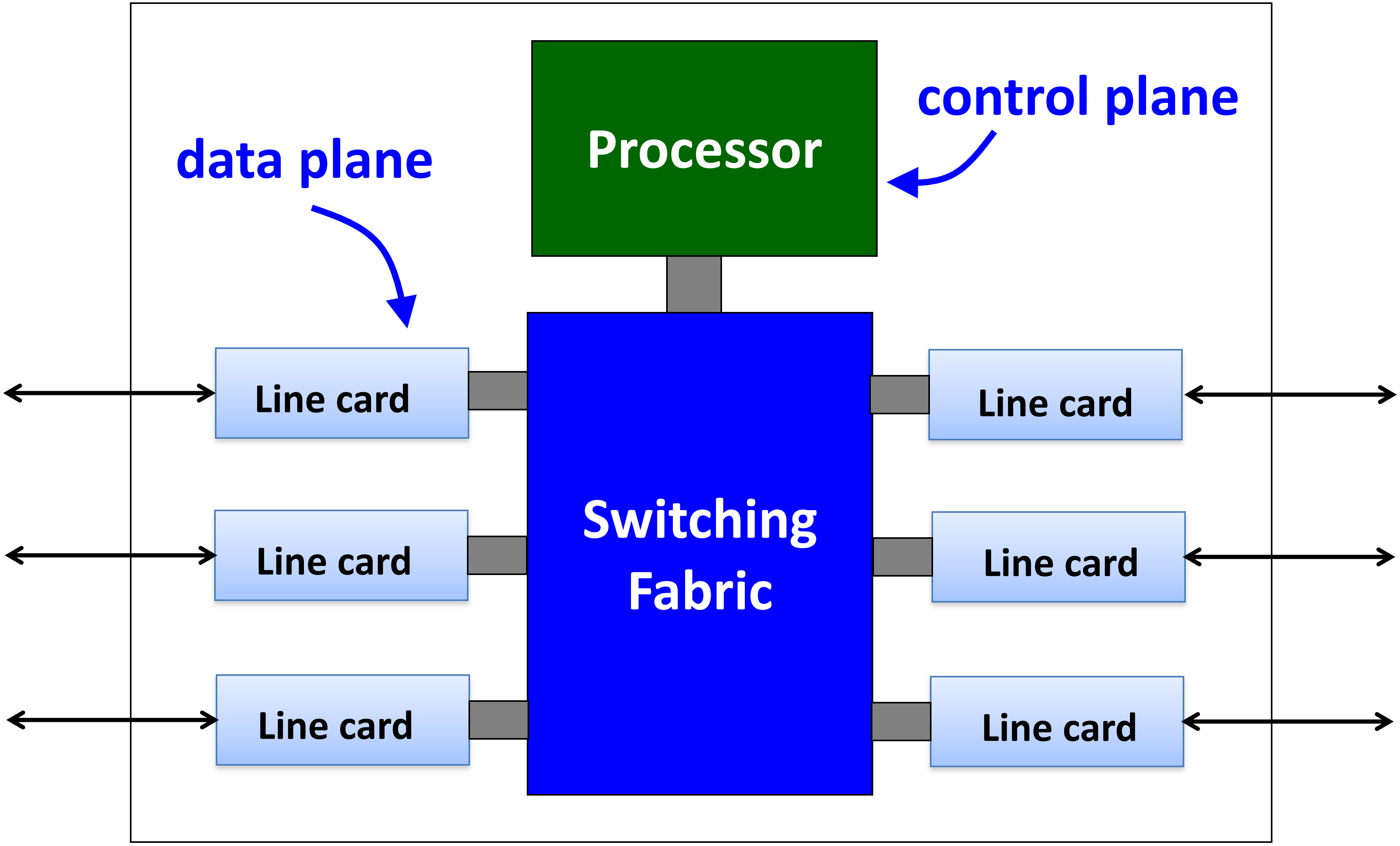
Today: Queue Management

What can individual *links* do to make good use of shared underlying resources?



Packet Queues

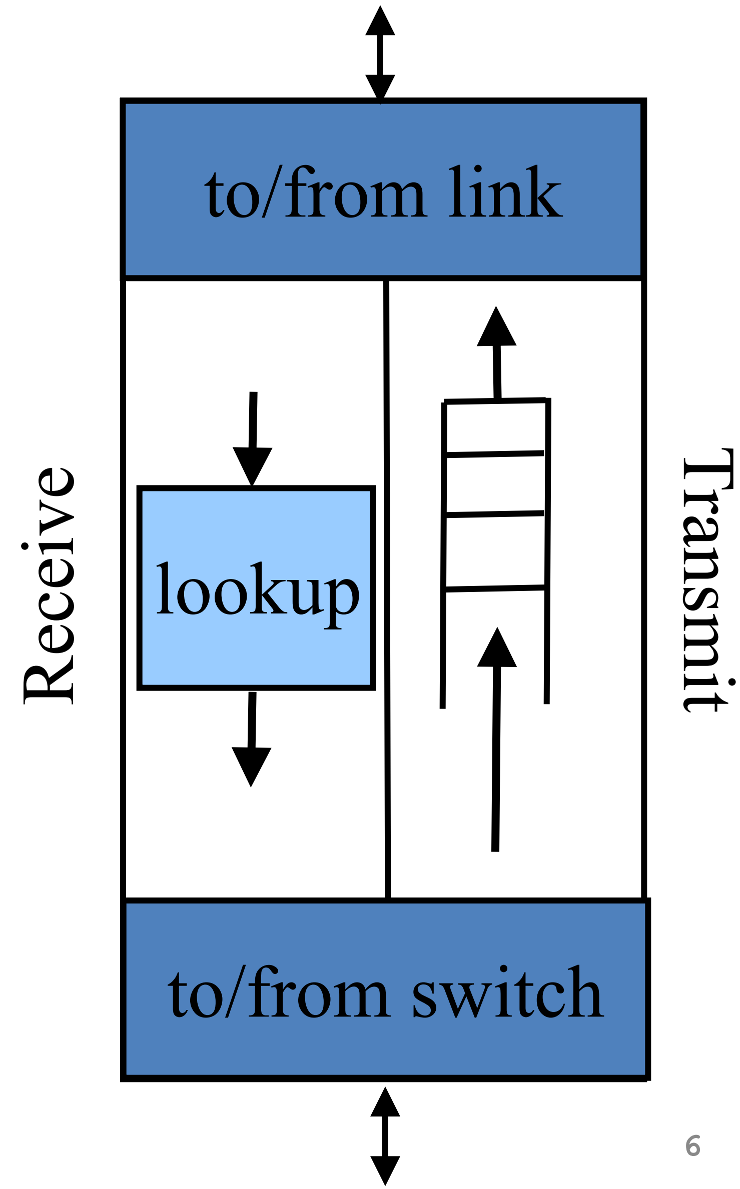
Router



Processor
control plane
data plane
Switching Fabric
Line card
Line card
Line card
Line card
Line card
Line card

Line Cards (Interface Cards, Adaptors)

- Packet handling
  - Packet forwarding
  - Buffer management
  - Link scheduling
  - Packet filtering
  - Rate limiting
  - Packet marking
  - Measurement

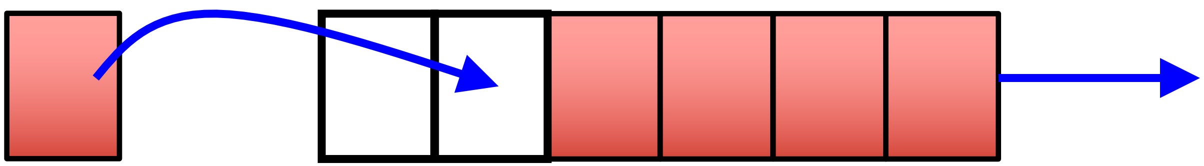


Queue Management Issues

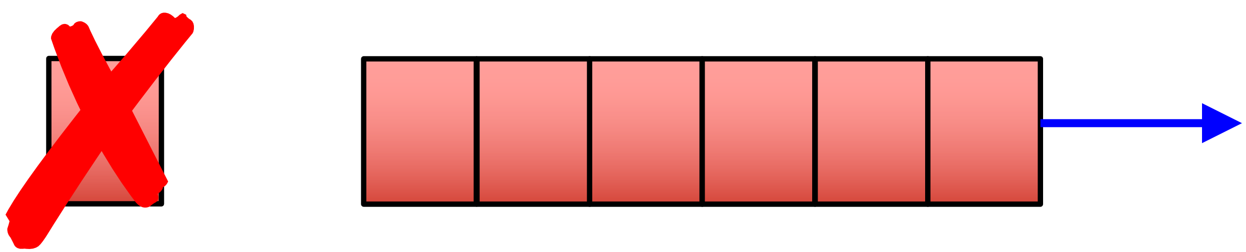
- Scheduling discipline
  - Which packet to send?
  - Some notion of fairness? Priority?
- Drop policy
  - When should you discard a packet?
  - Which packet to discard?
- Goal: balance throughput and delay
  - Huge buffers minimize drops, but add to queuing delay (thus higher RTT, longer slow start, …)

FIFO Scheduling and Drop-Tail

- Access to the bandwidth: first-in first-out queue
  - Packets only differentiated when they arrive



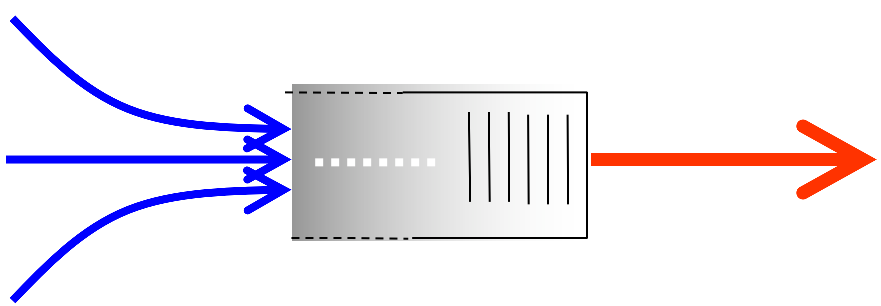
- Access to the buffer space: drop-tail queuing
  - If the queue is full, drop the incoming packet



Early Detection of Congestion

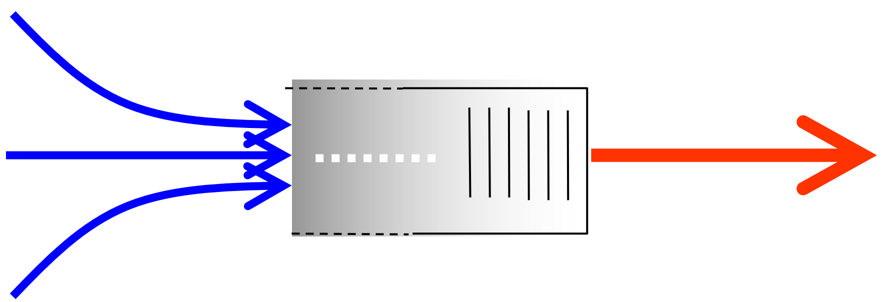
Bursty Loss From Drop-Tail Queuing

- TCP depends on packet loss
  - Packet loss is indication of congestion
  - TCP additive increase drives network into loss
- Drop-tail leads to *bursty* loss
  - *Congested link*: many packets encounter full queue
  - *Synchronization*: many connections lose packets at once



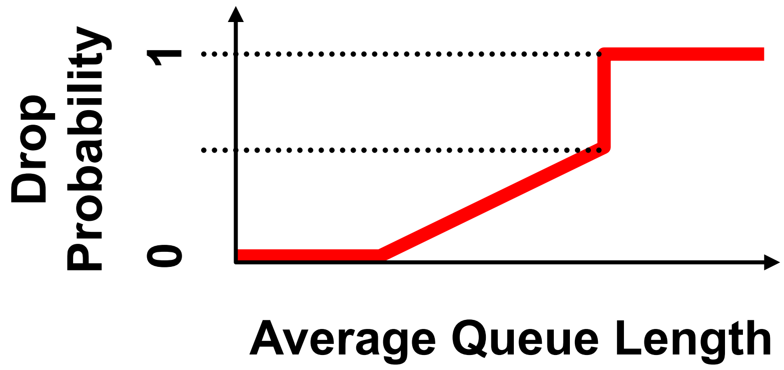
Slow Feedback from Drop Tail

- Feedback comes when buffer is completely full
  - … even though the buffer has been filling for a while
- Plus, the filling buffer is increasing RTT
  - … making detection even slower
- Better to give early feedback
  - Get 1-2 connections to slow down before it's too late!



Random Early Detection (RED)

- Router notices that queue is getting full
  - … and randomly drops packets to signal congestion
- Packet drop probability
  - Drop probability increases as queue length increases
  - Else, set drop probability *f*(avg queue length)



Properties of RED

- Drops packets before queue is full
  - In the hope of reducing the rates of some flows
- Tolerant of burstiness in the traffic
  - By basing the decisions on average queue length
- Which of the following are true?

  (Y) Drops packet in proportion to each flow's rate

  (M) High-rate flows selected more often

  (C) Helps desynchronize the TCP senders

  (A) All of the above

Properties of RED

- Drops packets before queue is full
  - In the hope of reducing the rates of some flows
- Tolerant of burstiness in the traffic
  - By basing the decisions on average queue length
- Which of the following are true?

  (Y) Drops packet in proportion to each flow's rate

  (M) High-rate flows selected more often

  (C) Helps desynchronize the TCP senders

  (A) All of the above

Problems With RED

- **Hard** to get **tunable parameters** just right
  - **How early** to start dropping packets?
  - **What slope** for increase in drop probability?
  - **What time scale** for averaging queue length?
- RED has **mixed adoption** in practice
  - If parameters aren't set right, RED doesn't help
- Many other variations in research community
  - Names like "Blue" (self-tuning), "FRED"…

From Loss to Notification

Feedback: From loss to **notification**

- Early dropping of packets (RED)
  - **Good:** gives early feedback
  - **Bad:** must **drop** the packet to give the feedback

- ***Explicit Congestion Notification (ECN)***
  - Router marks the packet with an ECN bit
  - Sending host interprets as a sign of congestion

Explicit Congestion Notification

- Needs support by router, sender, AND receiver
  – End-hosts check ECN-capable during TCP handshake
- ECN protocol (repurposes 4 header bits)
  1. **Sender** marks “ECN-capable” when sending
  2. If **router** sees “ECN-capable” and congested, marks packet as “ECN congestion experienced”
  3. If **receiver** sees “congestion experienced”, marks “ECN echo” flag in responses until congestion ACK’d
  4. If **sender** sees “ECN echo”, reduces *cwnd* and marks “congestion window reduced” flag in next packet

ECN Questions

Why separate ECN experienced and echo flags?

(Y) Detect reverse path congestion with “experienced”

(M) Congestion could happen in either direction, want sender to react to forward direction

(C) Both the above

ECN Questions

Why separate ECN experienced and echo flags?

(Y) Detect reverse path congestion with “experienced”

(M) Congestion could happen in either direction, want sender to react to forward direction

(C) Both the above

Why “echo” resent & “congestion window reduced” sender acknowledgement?

(Y) Congestion in reverse path can lose ECN-echo, still want to respond to congestion in forward path

(M) Only should apply backoff once per cwnd

(C) Both the above

ECN Questions

Why separate ECN experienced and echo flags?

(Y) Detect reverse path congestion with “experienced”

(M) Congestion could happen in either direction, want sender to react to forward direction

(C) Both the above

Why “echo” resent & “congestion window reduced” sender acknowledgement?

(Y) Congestion in reverse path can lose ECN-echo, still want to respond to congestion in forward path

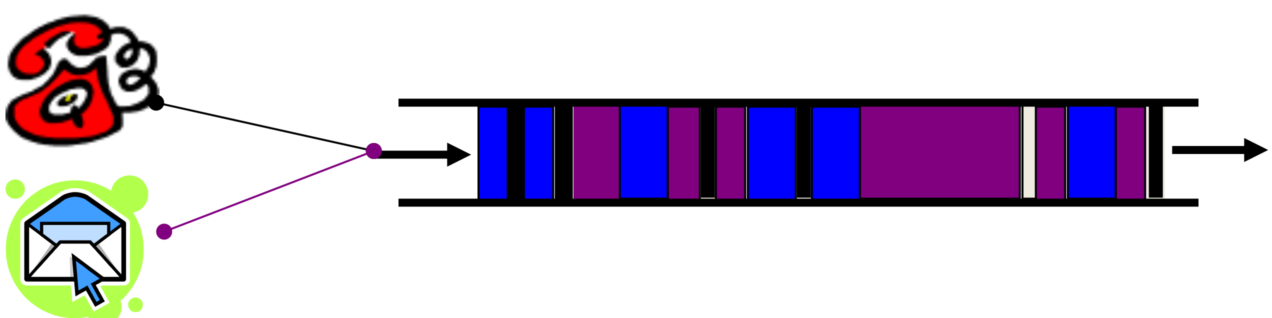
(M) Only should apply backoff once per cwnd

(C) Both the above

Link Scheduling

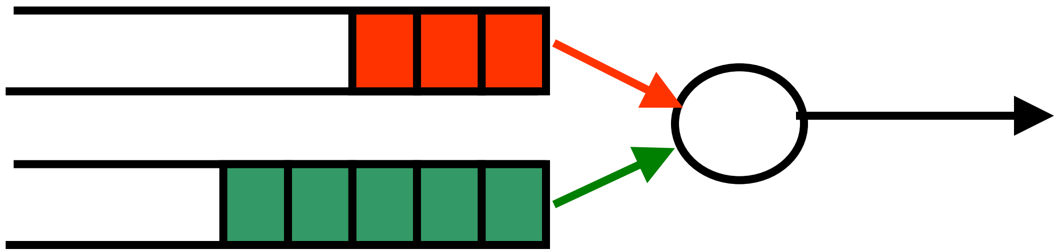
First-In First-Out Scheduling

- First-in first-out scheduling
  - Simple, but restrictive
- Example: two kinds of traffic
  - Voice over IP needs low delay
  - E-mail is not that sensitive about delay
- **Voice traffic waits behind e-mail**



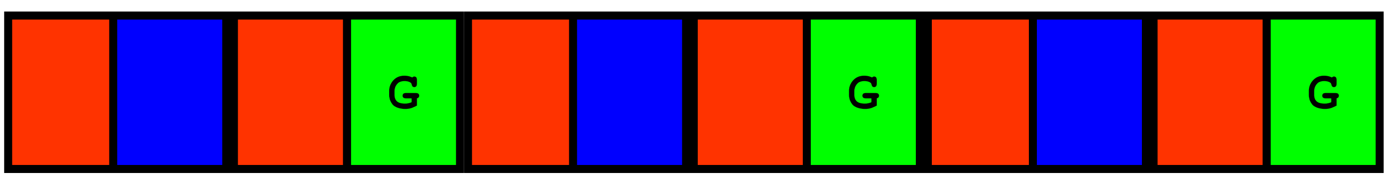
Strict Priority Queuing

- Multiple levels of ***priority***
  - **Always** transmit **high-priority** traffic (when present)
- **Isolation for** the **high-priority** traffic
  - Almost like it has a dedicated link
  - Except for (small) delay for packet transmission
- But, **lower-priority traffic may starve**



Weighted Fair Queuing

- Weighted fair queuing
  - Assign each queue a fraction of the link bandwidth
  - Rotate across queues on a small time scale



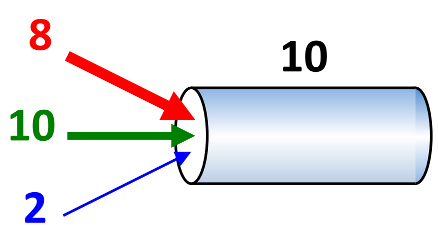
50% red, 25% blue, 25% green(G)

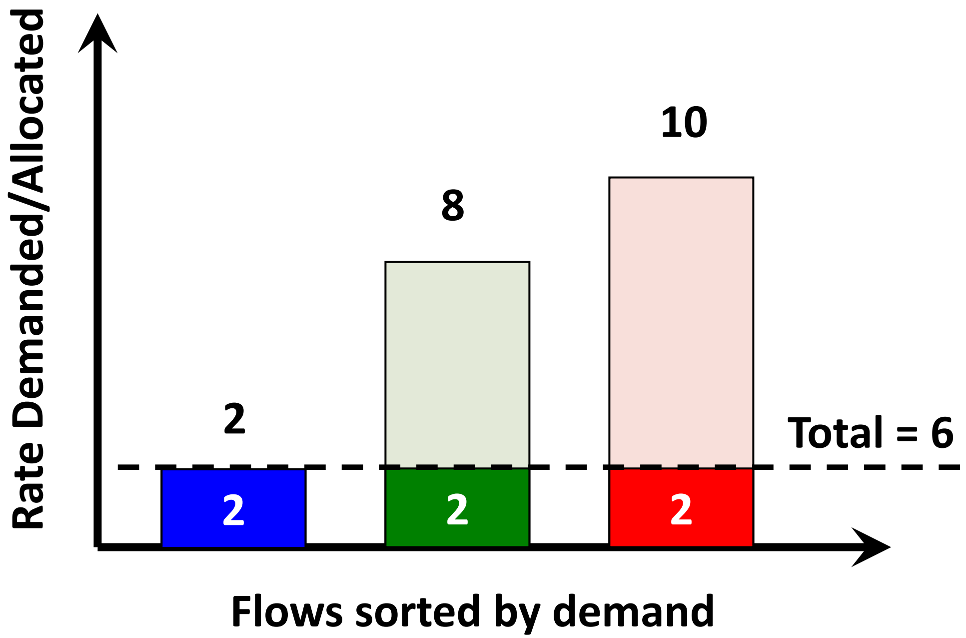
Weighted Fair Queuing

- If non-work conserving (resources can go idle)
  - Each flow gets at *most* its allocated weight
- WFQ is work-conserving
  - Send extra traffic from one queue if others are idle
  - Results in (Y) higher or (M) lower utilization than non-work conserving?
- WFQ results in max-min fairness
  - Maximize the minimum rate of each flow

Max-Min Fairness

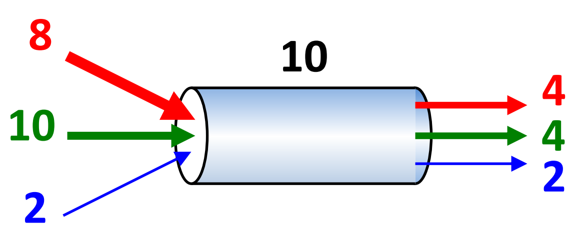
- Maximize the minimum rate of each flow
  1. Allocate in order of increasing demand
  2. No flow gets more than demand
  3. The excess, if any, is equally shared





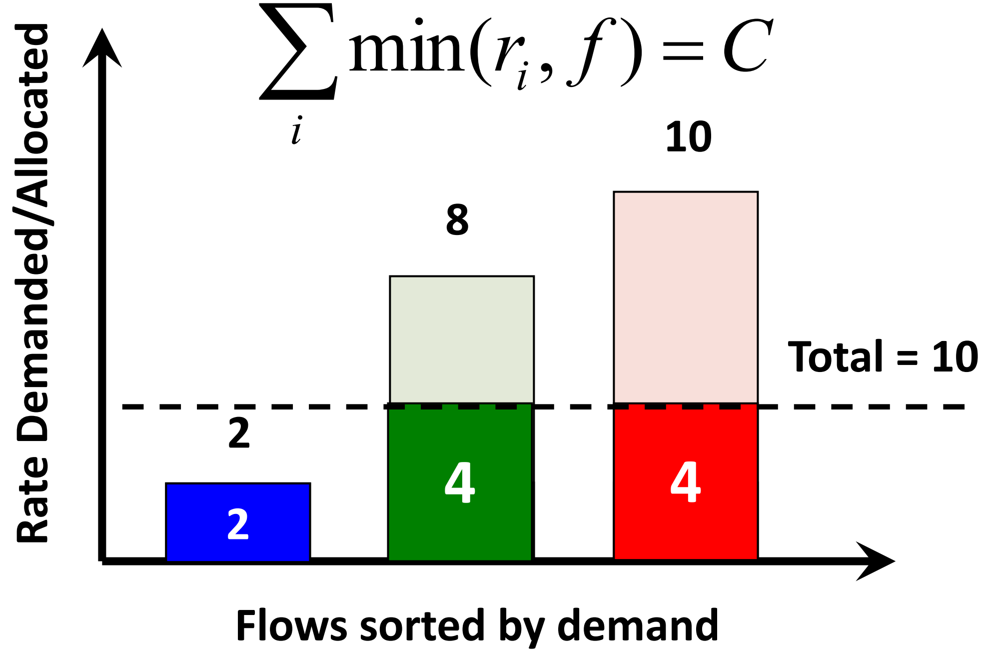
Max-Min Fairness

- Maximize the minimum rate of each flow
  1. Allocate in order of increasing demand
  2. No flow gets more than demand
  3. The excess, if any, is equally shared

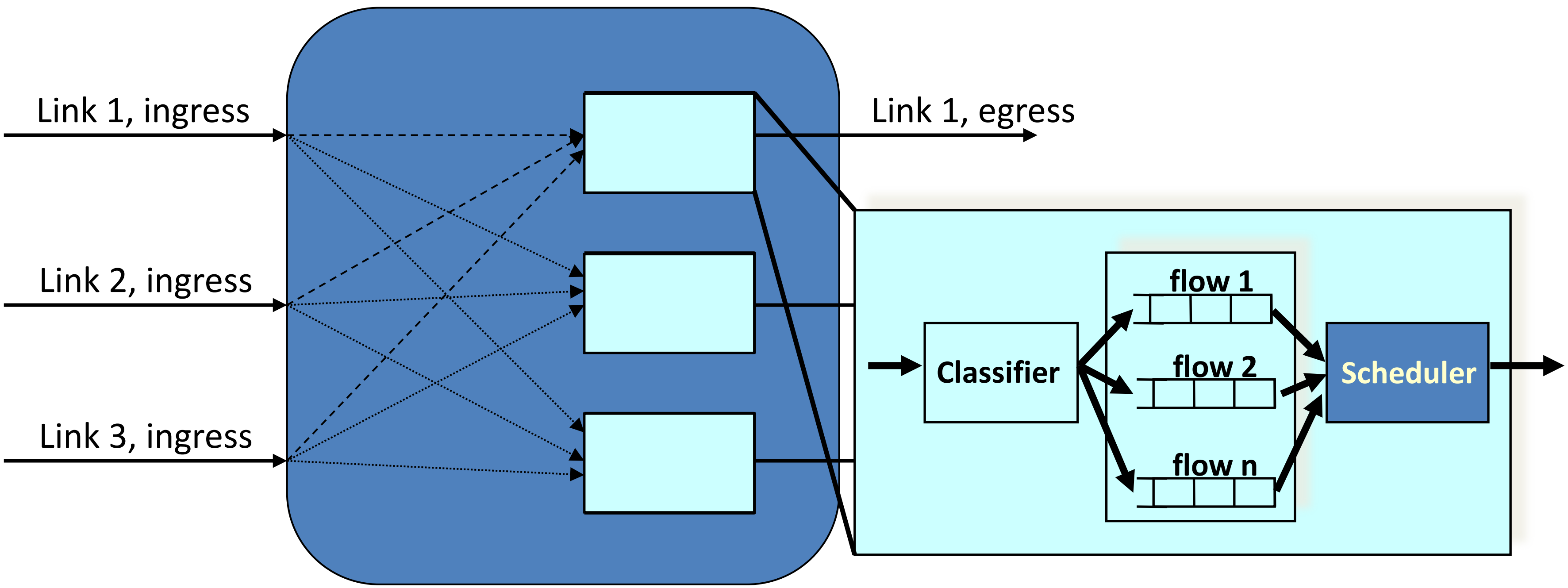


f = 4:
min(8, 4) = 4
min(10, 4) = 4
min(2, 4) = 2

$$\sum_i \min(r_i, f) = C$$



Weighted Fair Queuing



WFQ slides credit: Nick McKeown (Stanford)

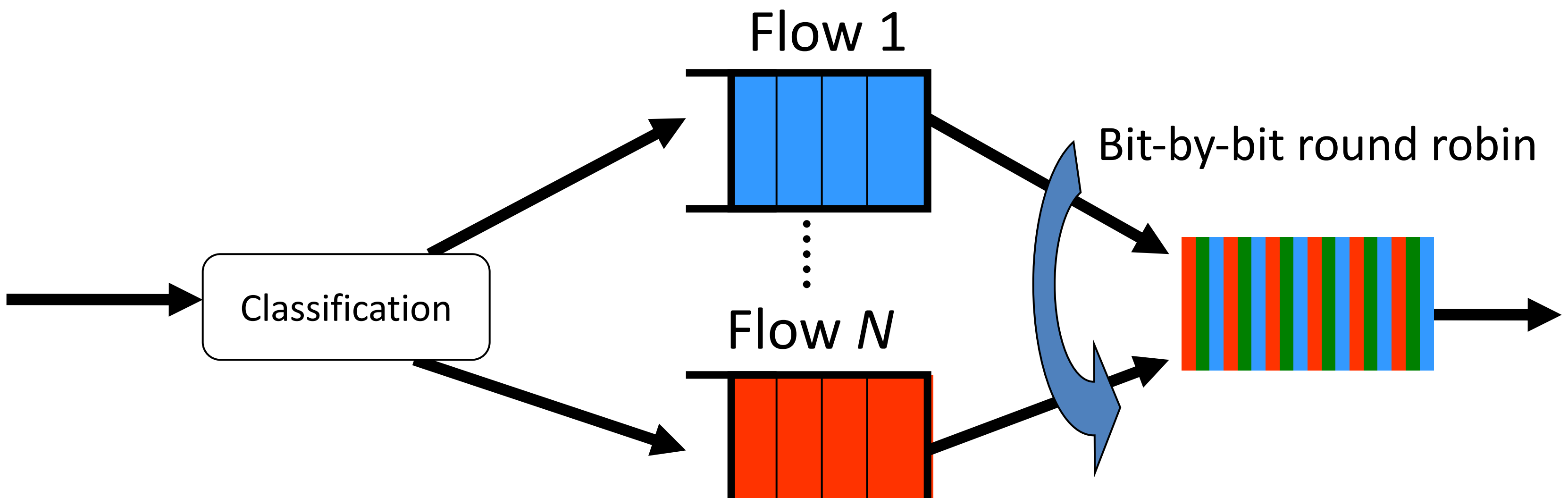
Bit-by-Bit Fair Queuing

Question: What is a "flow"?

Flow 5-tuple: protocol, IP source/dest, port src/dest

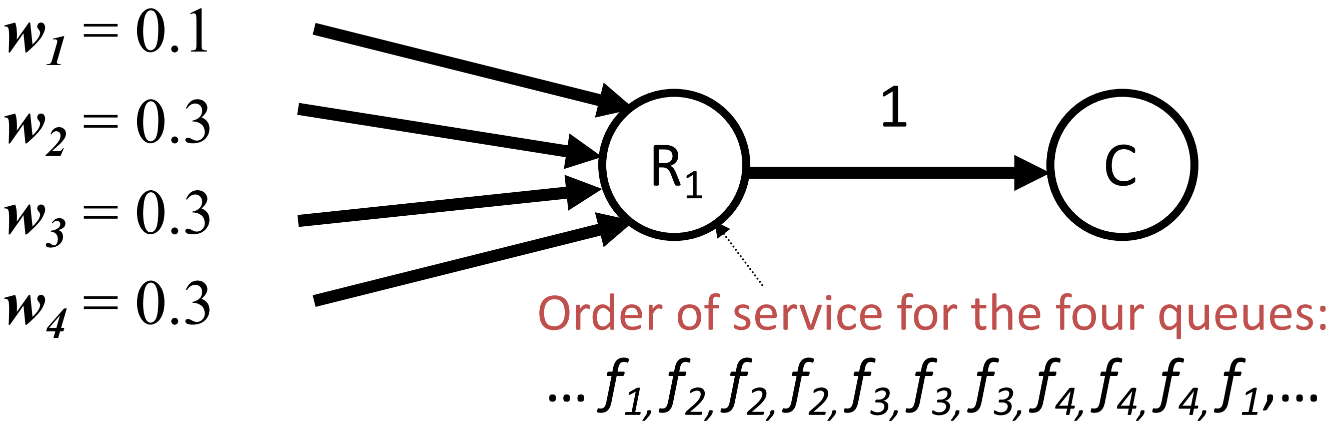
Question: How can we give weights?

Protocol class, bit markings, prefixes, etc.



Bit-by-Bit Weighted Fair Queuing

- Flows allocated different rates by servicing different number of bits for each flow during each round.



Packet vs. “Fluid” System

- Bit-by-bit FQ is not implementable:

  …In real packet-based systems:
  - One queue is served at any given time
  - Packet transmission cannot be preempted

- Goal: A packet scheme close to fluid system
  - Bound performance w.r.t. fluid system

First Cut: Simple Round Robin

- Serve a packet from non-empty queues in turn
  - Let's assume all flows have equal weight
- Variable packet length → get more service by sending bigger packets
- Unfair instantaneous service rate (especially with variable weights)
  - What if a packet arrives right after its "turn"?

Packet-by-packet Fair Queuing (Weighted Fair Queuing)

Copes better with variable size packets and weights

Key Idea:

1. Determine the *finish time* of packets in bit-by-bit system, assuming no more arrivals
2. Serve packets in order of finish times

Implementing WFQ

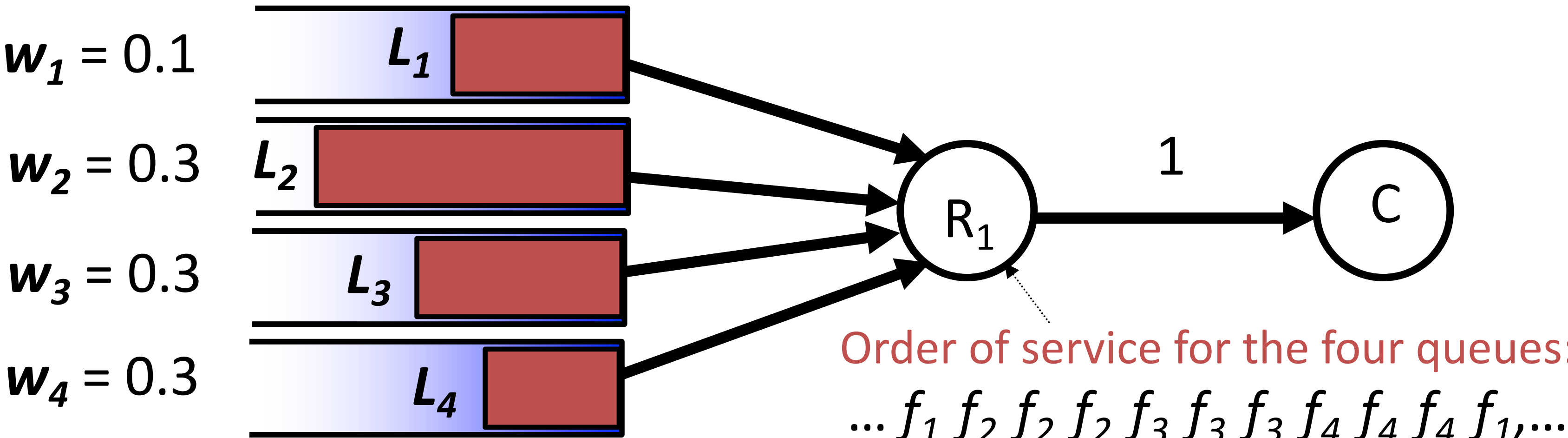
Challenge: Determining finish time is hard

Idea: Don’t need finish time. Need finish **order.**

The finish order is a lot easier to calculate.

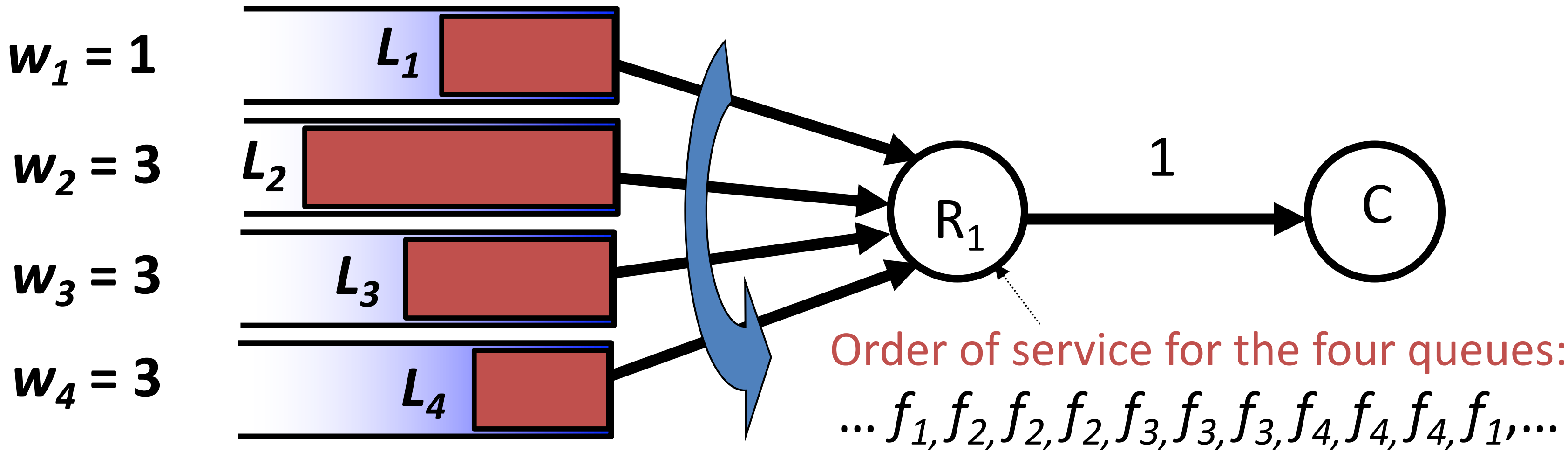
Finish order

In what order do the packets finish? **Increasing L_i/w_i**



Does not change with future packet arrivals!

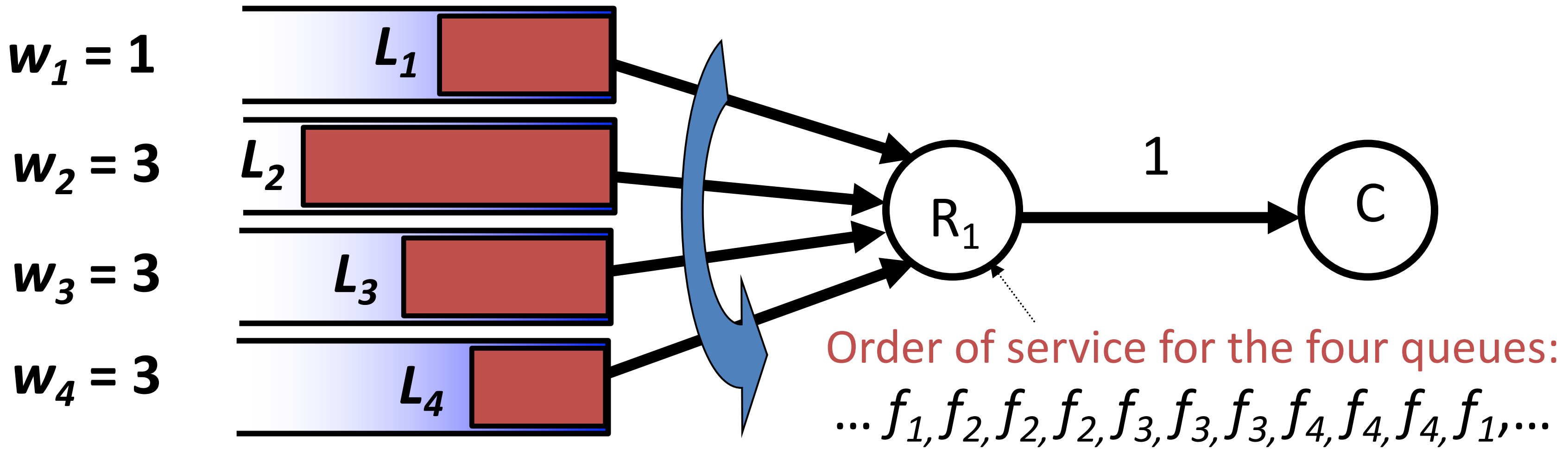
Bit-by-bit System Round



Round – One complete cycle through all the queues sending w_i bits per queue

Question: How long does a round take?

Bit-by-bit System Round



Round – One complete cycle through all the queues sending w_i bits per queue

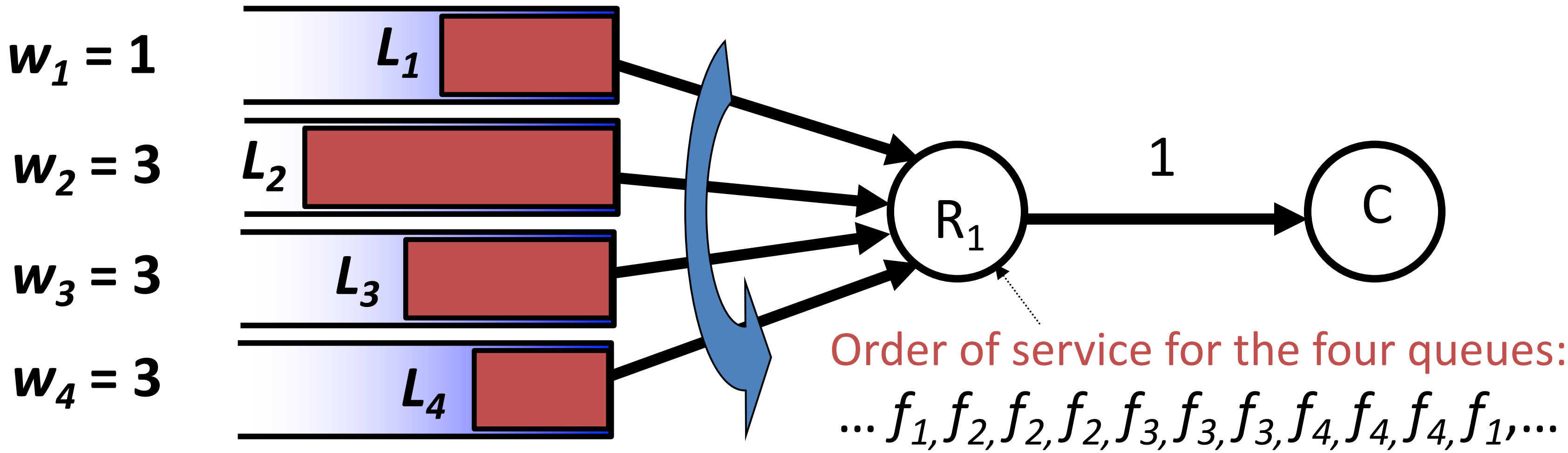
$$\frac{dR}{dt} = \frac{C}{\sum_{j \in B(t)} w_j}$$

R(t) **= # of rounds at time t**

C **= link rate**

B(t) **= set of backlogged flows**

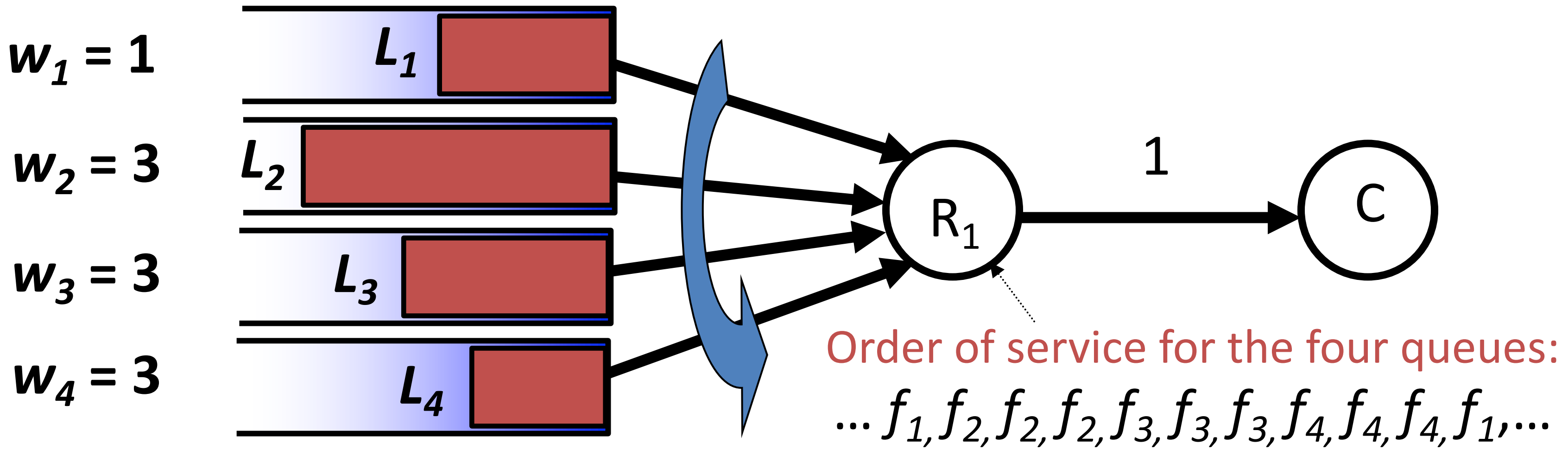
Bit-by-bit System Round



Round – One complete cycle through all the queues sending w_i bits per queue

Question: How many rounds does it take to serve a packet of length L from flow i?

Bit-by-bit System Round



Round – One complete cycle through all the queues sending w_i bits per queue

Packet of length L takes L/w_i rounds to serve

Round (aka. "Virtual Time") Implementation of WFQ

Question: What is *finish* round of k^{th} packet – F_i^k?

Round (aka "Virtual Time") Implementation of WFQ

Assign a start/finish round to each packet at arrival
→ serve packets in order of finish rounds

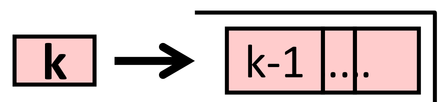
Suppose k^{th} packet of flow i arrives at time a

Start round of k^{th} *packet*

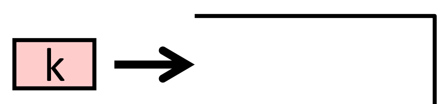
Finish round of $(k\text{-}1)^{st}$ *packet*

Round number at time a

$$S_i^k = \begin{cases} F_i^{k-1} \\ R(a) \end{cases}$$



(Flow i backlogged)



(Flow i empty)

Putting it All Together

For k^{th} packet of flow *i* arriving at time *a:*

$$S_i^k = \max(F_i^{k-1}, R(a))$$

$$F_i^k = S_i^k + \frac{L_i^k}{w_i}$$

Question: How to compute R(a)?

$$\frac{dR}{dt} = \frac{C}{\sum_{j \in B(t)} w_j}$$

Simple approximation:

Set R(a) to start or finish round of packet currently in service

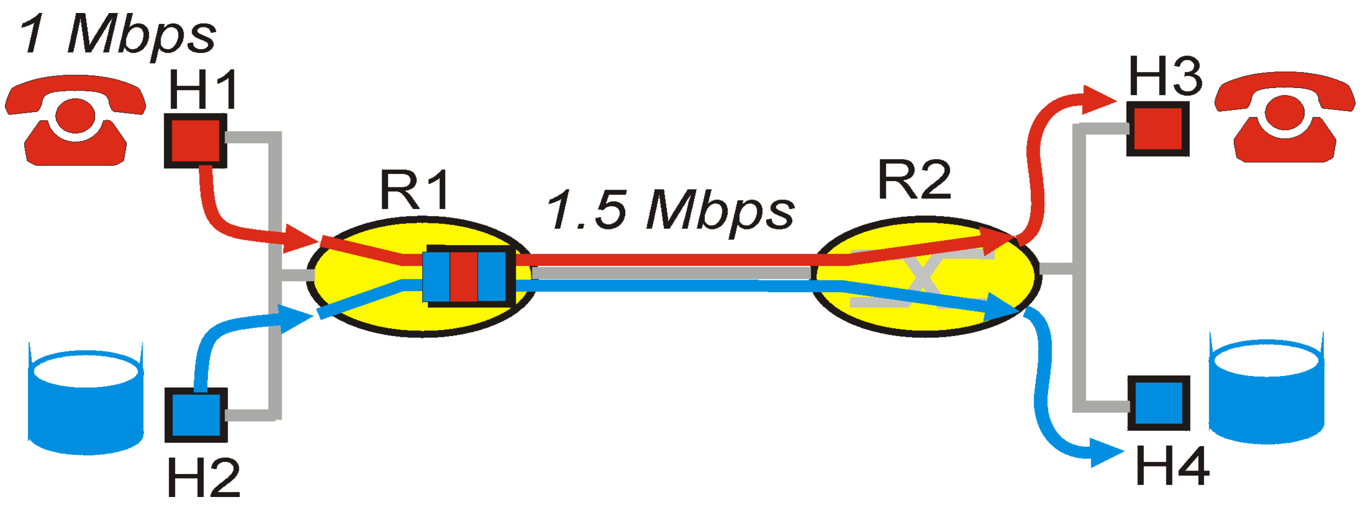
Implementation Trade-Offs

- FIFO
  - One queue, trivial scheduler
- Strict priority
  - One queue per priority level, simple scheduler
- Weighted fair scheduling
  - One queue per class, and more complex scheduler

Quality of Service Guarantees

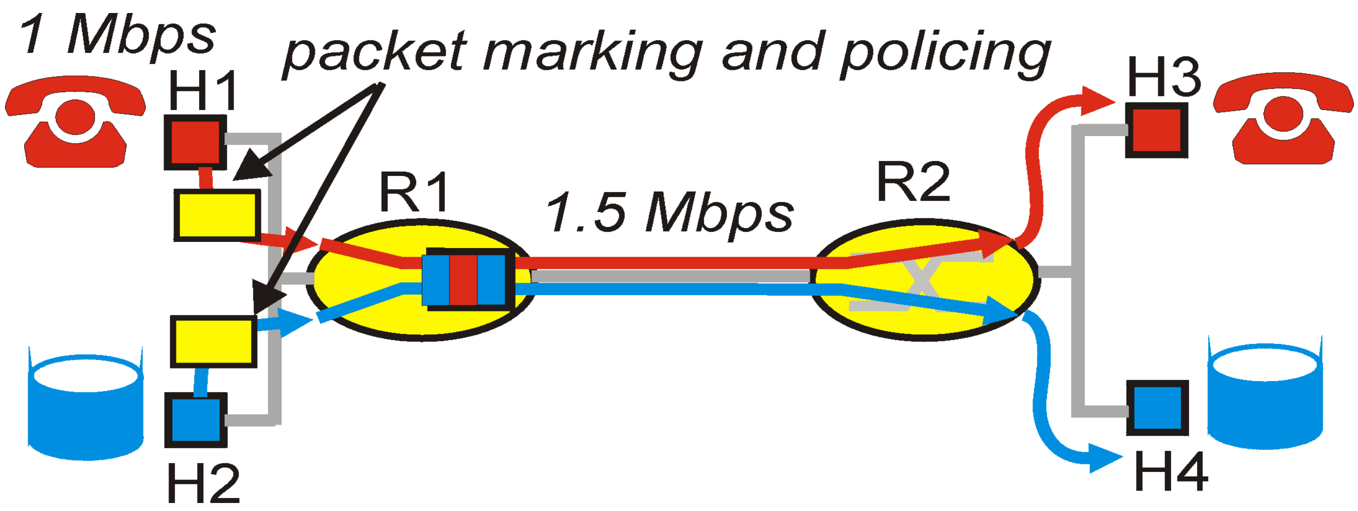
Distinguishing Traffic

- Applications compete for bandwidth
  - E-mail traffic can cause congestion/losses for VoIP
- Principle 1: Packet marking
  - So router can distinguish between classes
  - E.g., Type of Service (ToS) bits in IP header



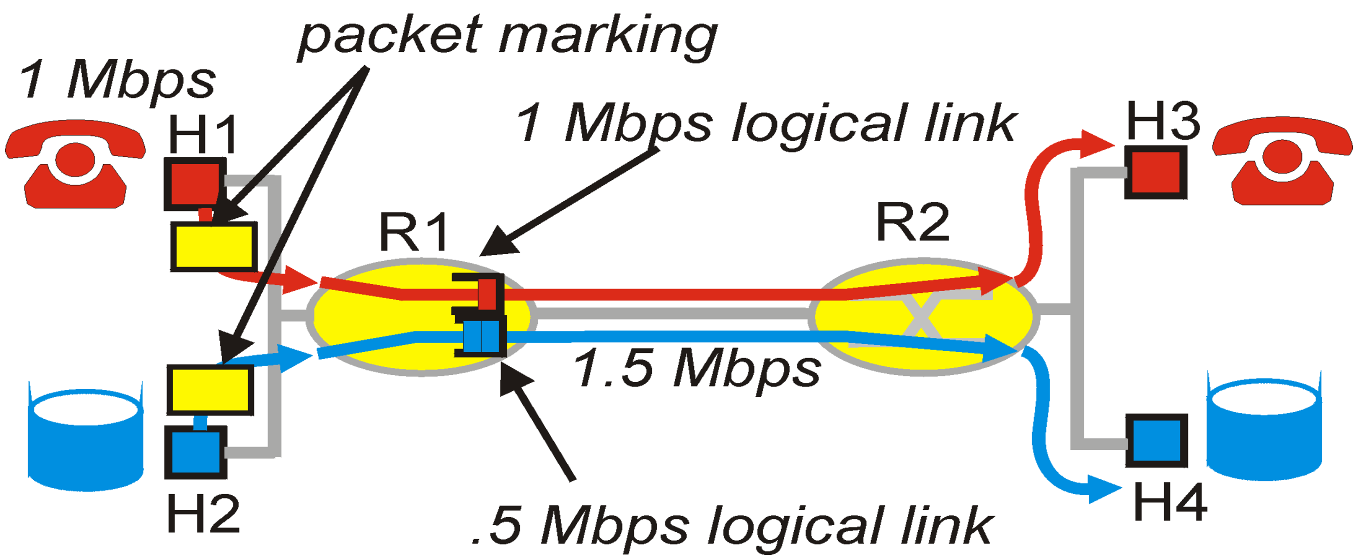
Preventing Misbehavior

- Applications misbehave
  - VoIP sends packets faster than 1 Mbps
- Principle 2: Policing
  - Protect one traffic class from another
  - By enforcing a rate limit on the traffic



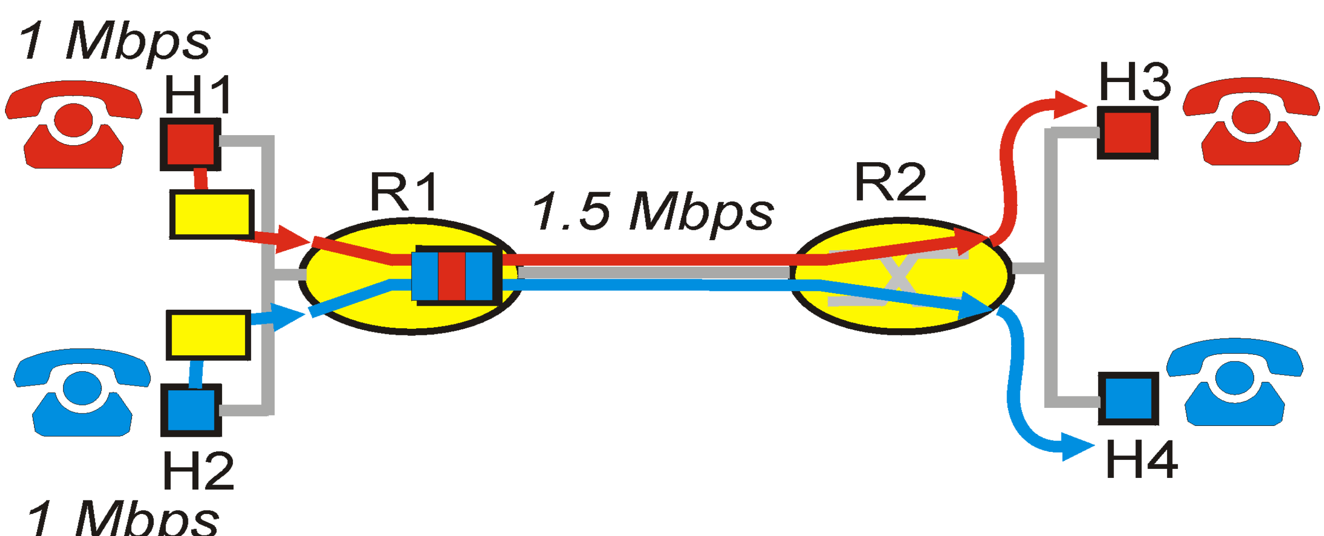
Subdividing Link Resources

- Principle 3: Link scheduling
  - Ensure each application gets its share
  - … while (optionally) using any extra bandwidth
  - E.g., weighted fair queuing



Reserving Resources, and Saying No

- Traffic cannot exceed link capacity
  - Deny access, rather than degrade performance
- Principle 4: Admission control
  - Application declares its needs in advance
  - Application denied if insufficient resources available



Quality of Service (QoS)

- Guaranteed performance
  - Alternative to best-effort delivery model
- QoS protocols and mechanisms
  - Packet classification and marking
  - Traffic shaping
  - Link scheduling
  - Resource reservation and admission control
  - Identifying paths with sufficient resources