Resource Allocation in Networks

Resource allocation in networks

- Very much like a resource allocation problem in operating systems
- How is it different?
  - Resources and jobs are different
  - Resources are buffers and link bandwidth
  - Jobs are flows
- CPU scheduling in OS → Packet scheduling in networks

Resource allocation..

- We can categorize resource allocation approaches based on implementation strategies in different ways:
  - Router based versus Host based
  - Feedback based versus Reservation based
  - Window based versus Rate based

Resource allocation…

- Several approaches presented in the literature:
  - Best effort: no commitment about QoS
  - Better than best effort: services make no deterministic guarantees about delay, but make a best effort for supporting QoS
  - Guaranteed throughput: provides strict QoS compliance
  - Bounded delay jitter: service guarantees upper and lower bounds on observed packet delays

Granularity of Resource Allocation

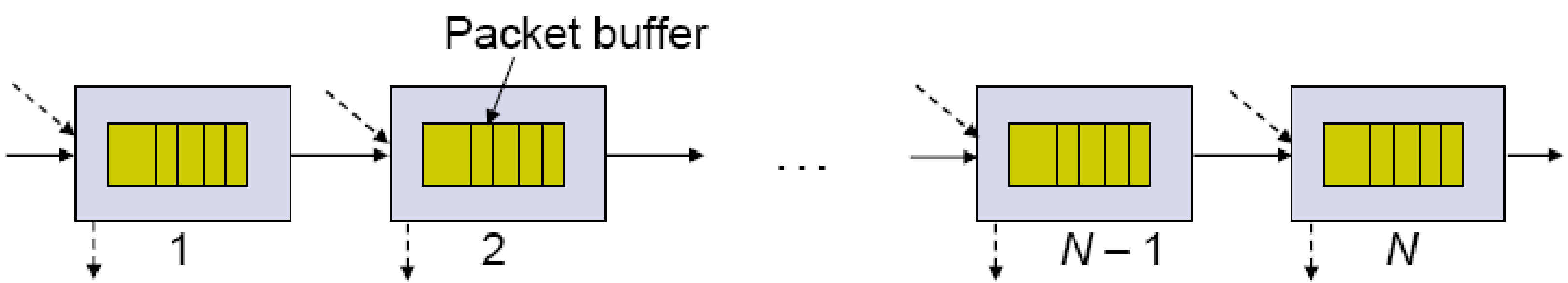
- Based on the management granularity, we can classify the approaches into three classes:
  - Packet level
  - Flow level
  - Flow aggregate level

Packet Level Resource Allocation

- This is mainly concerned with packet queuing, and packet scheduling at switches, routers, etc.
- ***Objective***: provide different treatments at the packet level so that some flows (applications) will receive better service

QoS Concerns with Packet Scheduling

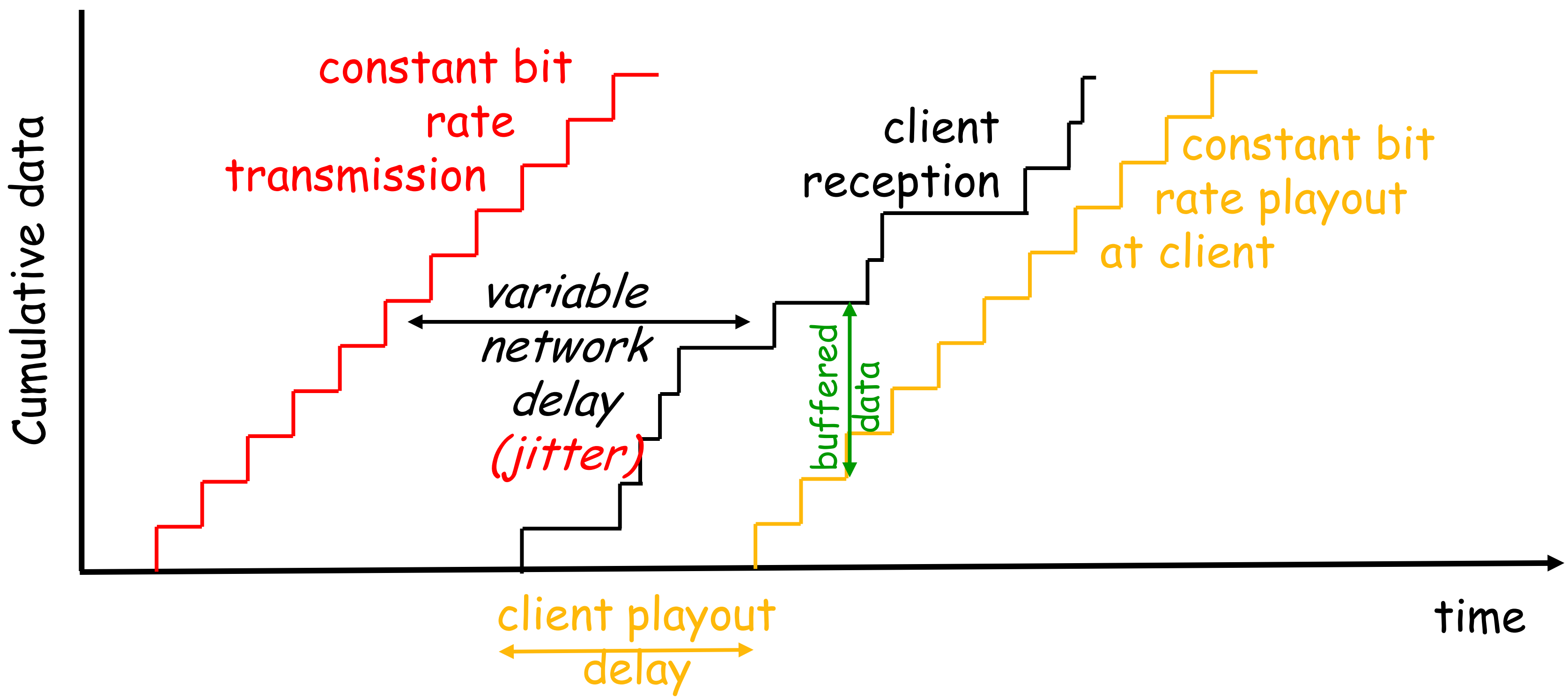
- End-to-end delay is the sum of all the per hop delays



- End-to-end delay can be bounded by upper bounding the delay at each hop

QoS concerns..

- Jitter – is the variability of delay. It is a major concern as well. Why?

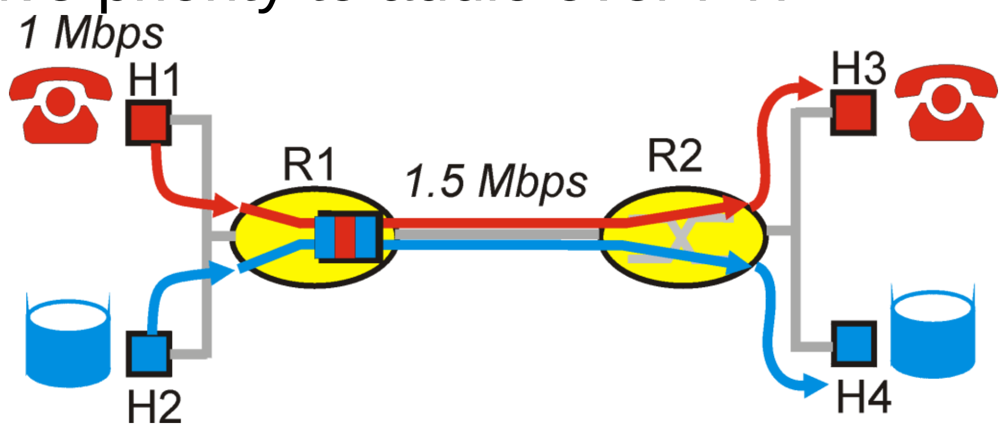


QoS concerns..

- Packet loss: happens when there is no more room in the buffers
- Causes for packet loss:
  - Surge in packet input rate
  - Congestion downstream

Principles for QOS Guarantees

- Example: 1Mbps IP phone, FTP share 1.5 Mbps link.
  - bursts of FTP can congest router, cause audio loss
  - want to give priority to audio over FTP

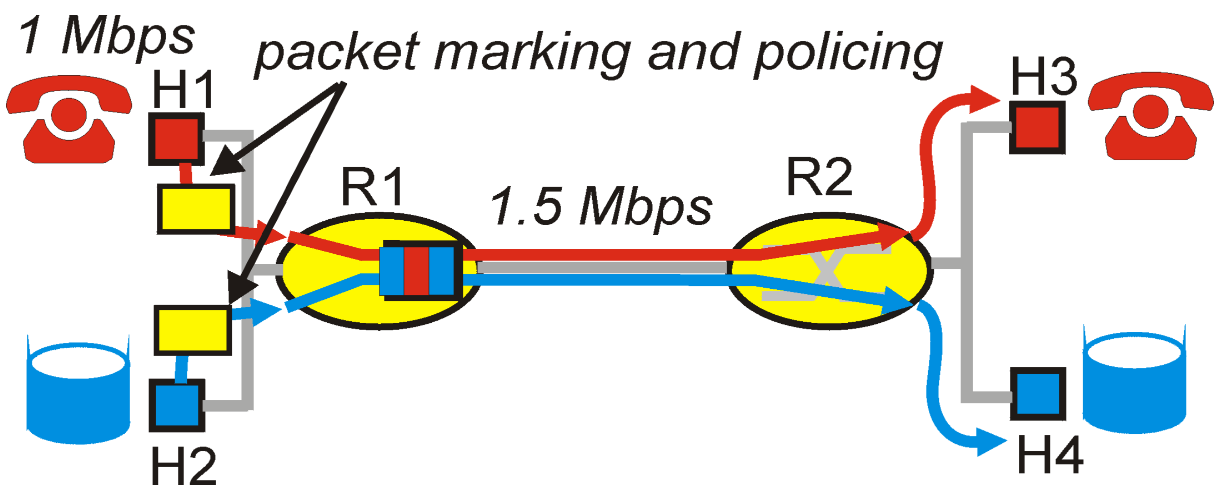


Principle 1

packet marking needed for router to distinguish between different classes; and new router policy to treat packets accordingly

Principles for QOS Guarantees (more)

- what if applications misbehave (audio sends higher than declared rate)
  - policing: force source adherence to bandwidth allocations
- marking and policing at network edge:

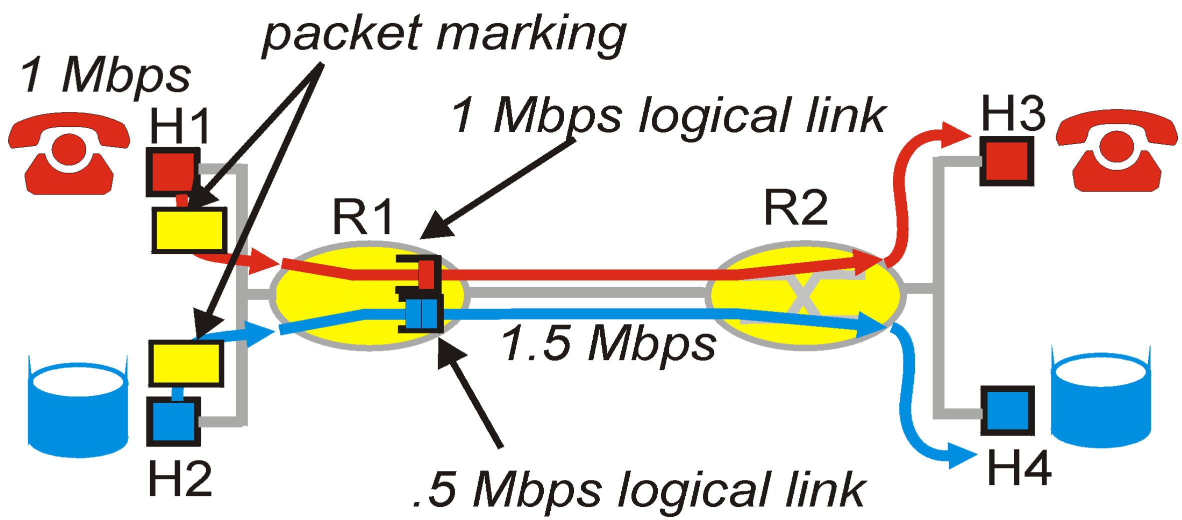


Principle 2

provide protection (*isolation*) for one class from others

Principles for QOS Guarantees (more)

- Allocating *fixed* (non-sharable) bandwidth to flow: *inefficient* use of bandwidth if flows doesn't use its allocation

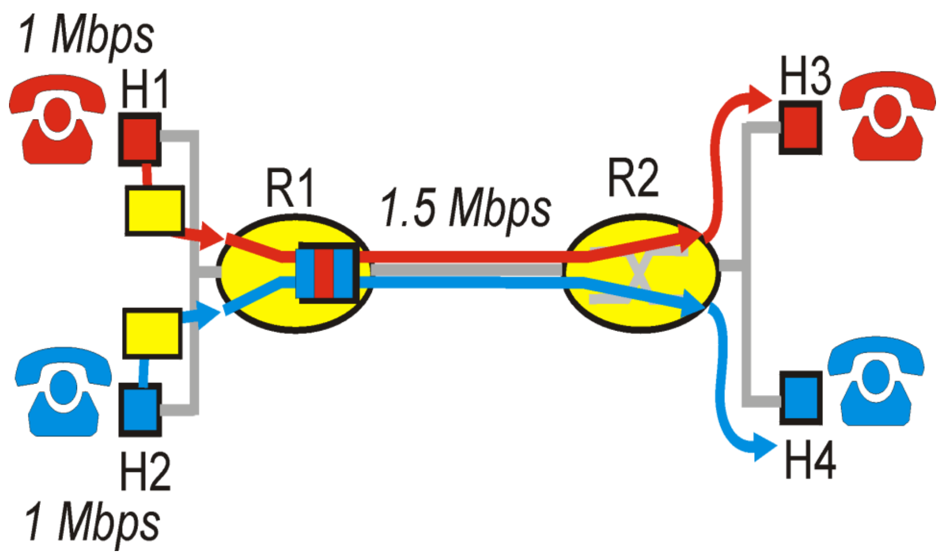


Principle 3

While providing isolation, it is desirable to use resources as efficiently as possible

Principles for QOS Guarantees (more)

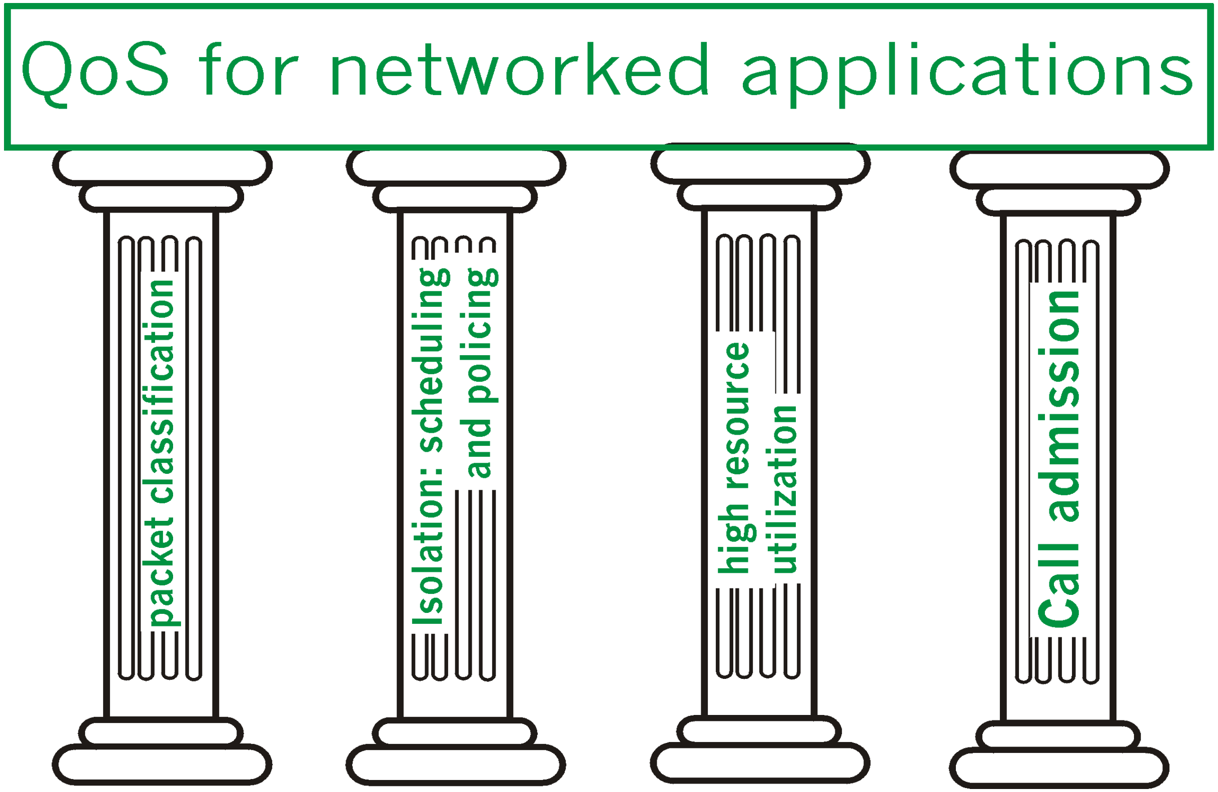
- *Basic fact of life:* can not support traffic demands beyond link capacity



Principle 4

Call Admission: flow declares its needs, network may block call (e.g., busy signal) if it cannot meet needs

Summary of QoS Principles



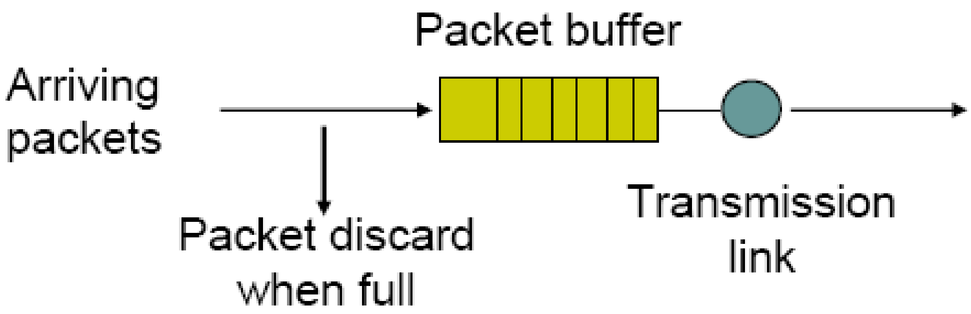
Let's next look at mechanisms for achieving this ....

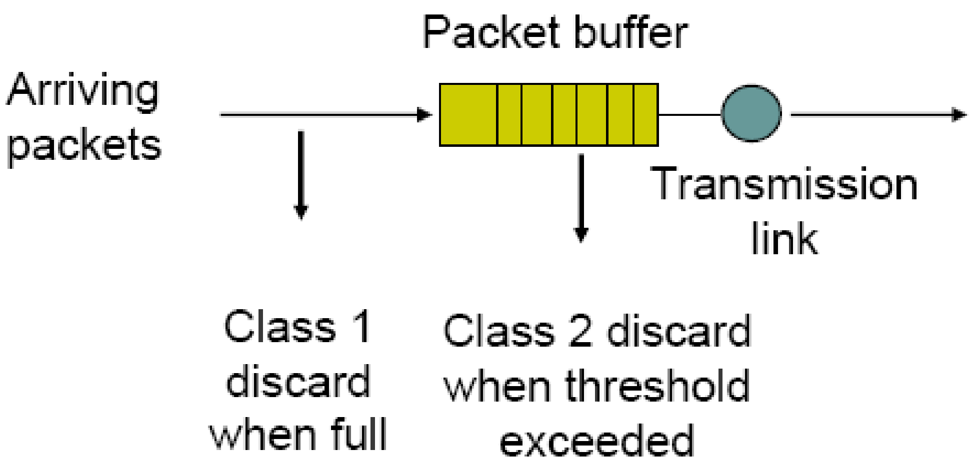
Queuing Disciplines

- Queuing algorithms allocate three nearly independent quantities:
  - bandwidth (which packets get transmitted)
  - promptness (when packets get transmitted)
  - buffer space (which packets are discarded by the gateway)

Queuing Disciplines

- Simplest queuing algo.:
  - FCFS (first come first serve)
  - order of arrival determines the bandwidth, promptness, and buffer space allocations
  - congestion control relegated to the sources





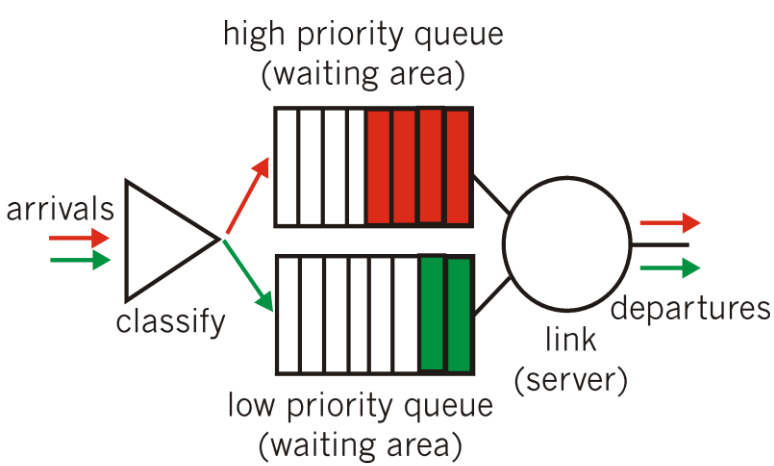
Queuing Disciplines

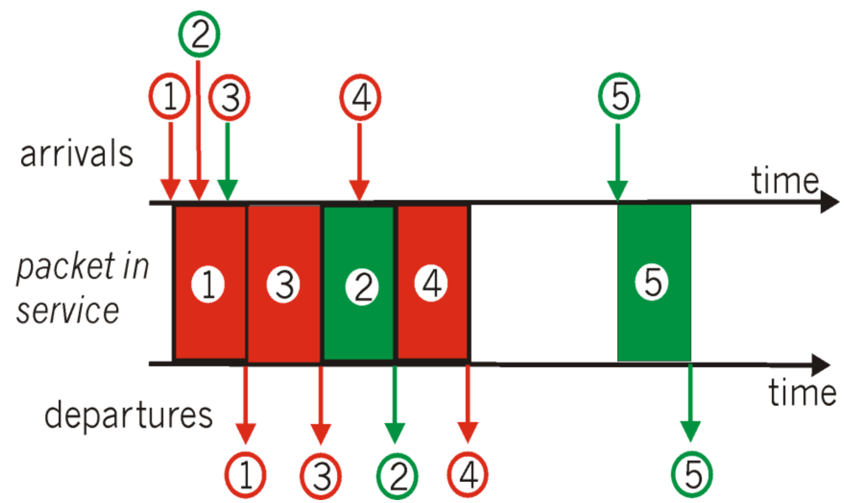
- FIFO with tail drop
  - use FIFO scheme
  - when the buffer space is full, drop the next packet that arrives at the router
- Problem with FCFS:
  - single source sending traffic at an arbitrarily high rate captures a good portion of the output bandwidth
  - congestion control may not be fair with ill-behaved sources

Queuing Disciplines: more

Priority scheduling: transmit highest priority queued packet

- multiple *classes*, with different priorities
  - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc..
  - Real world example?

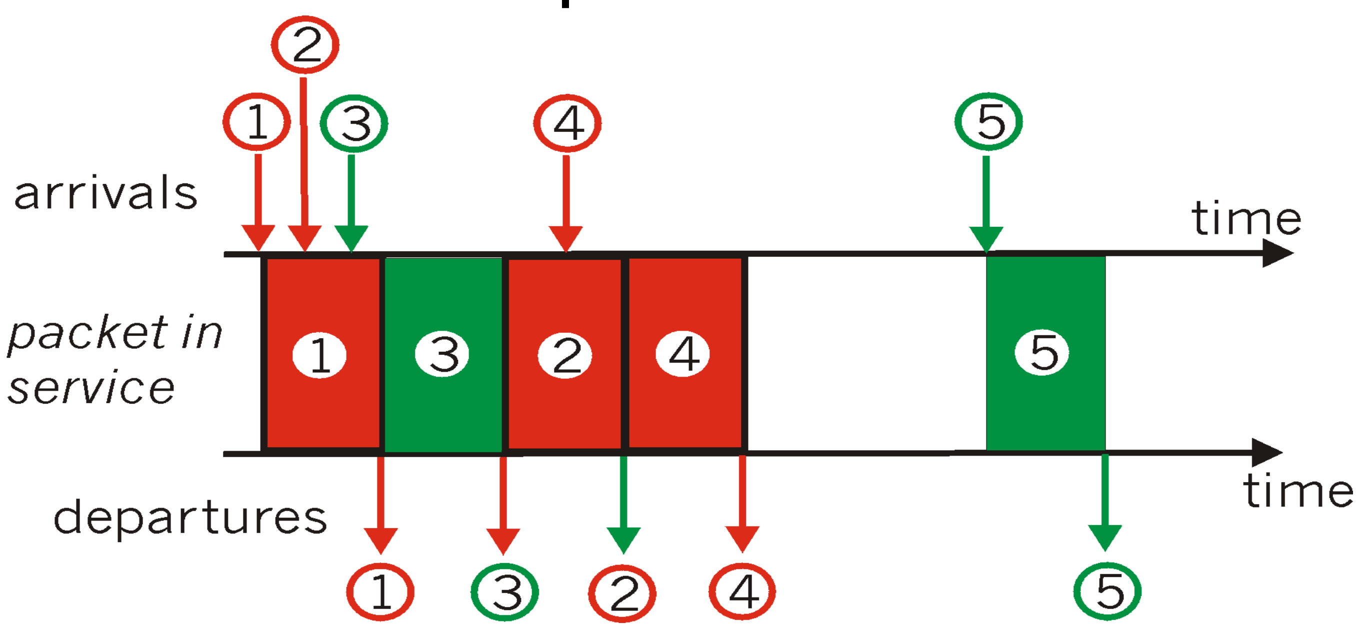




Queuing Discipline: still more

round robin scheduling:

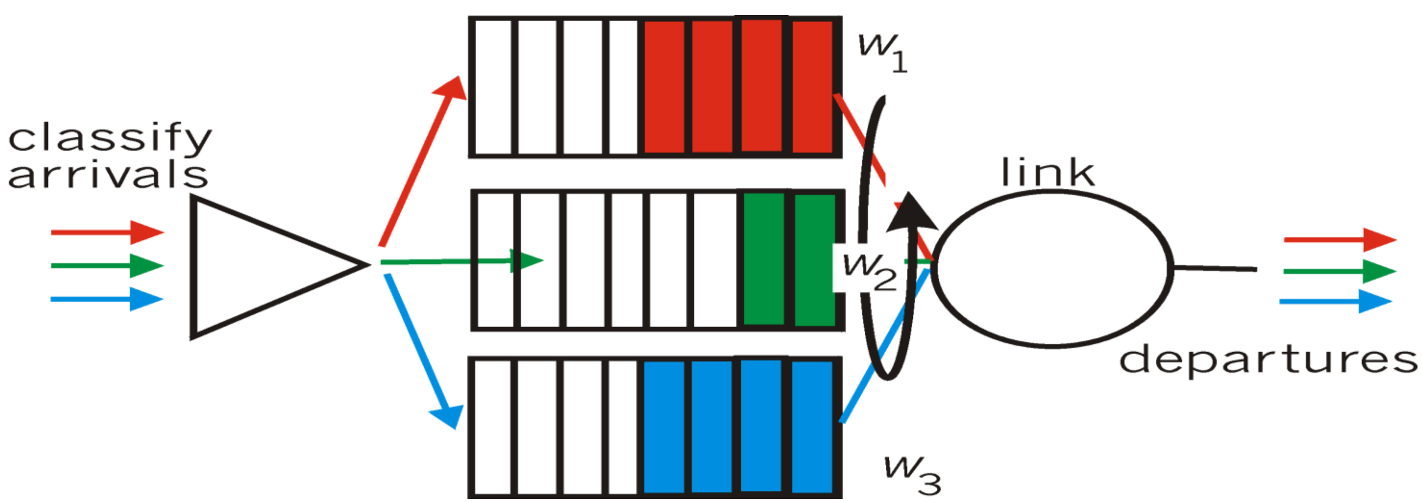
- multiple classes
- cyclically scan class queues, serving one from each class (if available)
- real world example?



Scheduling Policies: still more

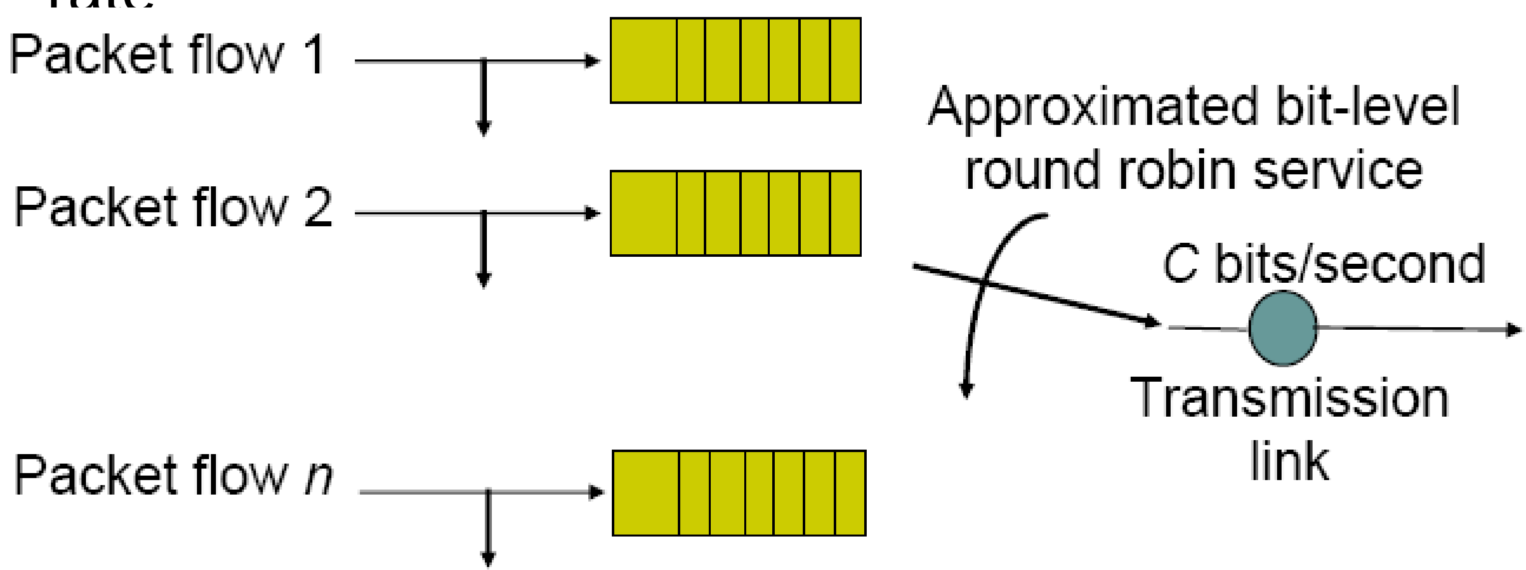
Weighted Fair Queuing:

- generalized Round Robin
- each class gets weighted amount of service in each cycle
- real-world example?



Fair Queuing

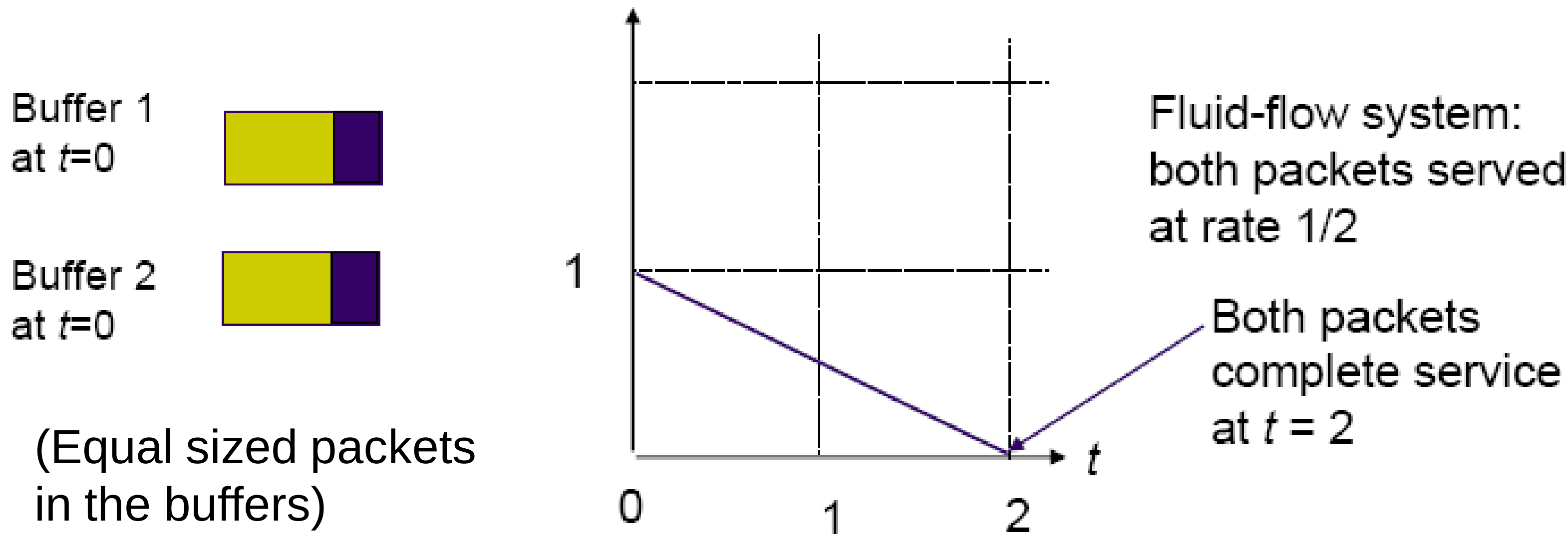
- Maintain a separate queue for each flow
- Service the queues in a round-robin fashion
- when queue reaches a particular length, additional packets for the flow are discarded -- flow cannot increase its share of the bandwidth by increasing flow rate

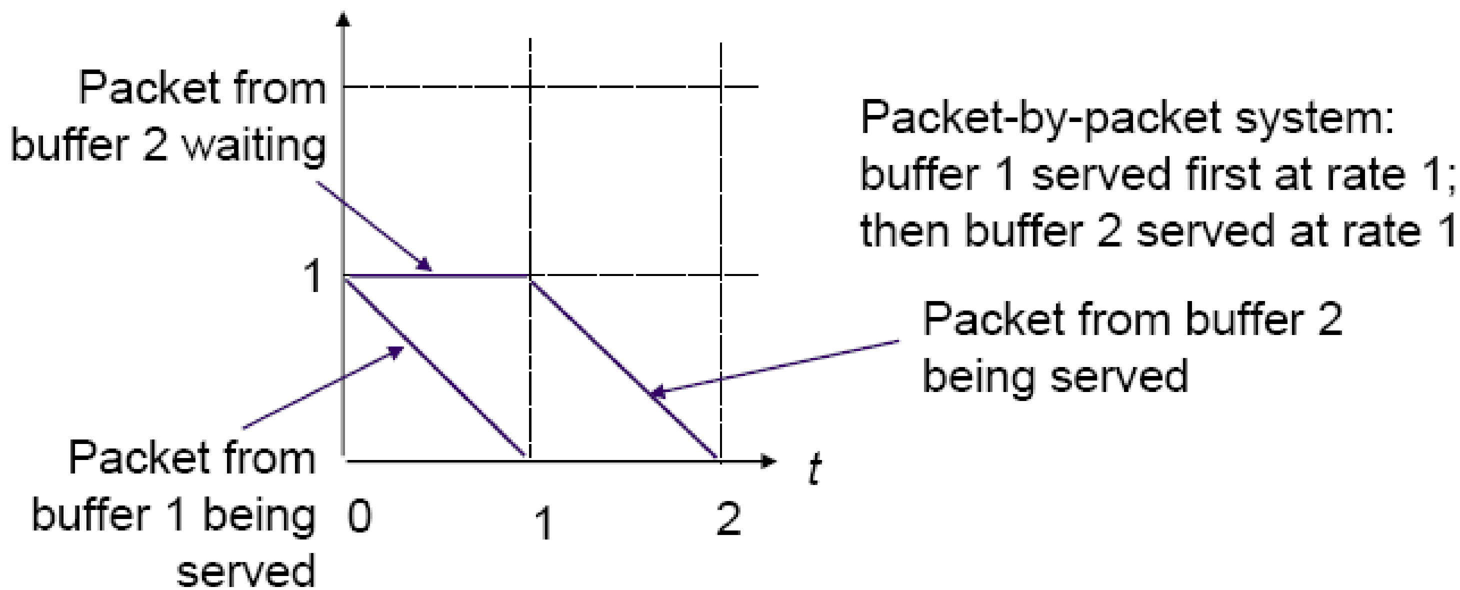


Queuing Disciplines

- Pure allocation of round-robin service
  - provides a fair allocation of packets-sent
  - due to varying packet sizes, does not guarantee fair allocation of bandwidth
- Bit-by-bit round-robin (BR)
  - allocates bandwidth fairly
  - not very practical -- only a hypothetical scheme

Bit-by-Bit Vs. Packet-by-Packet

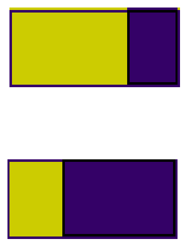




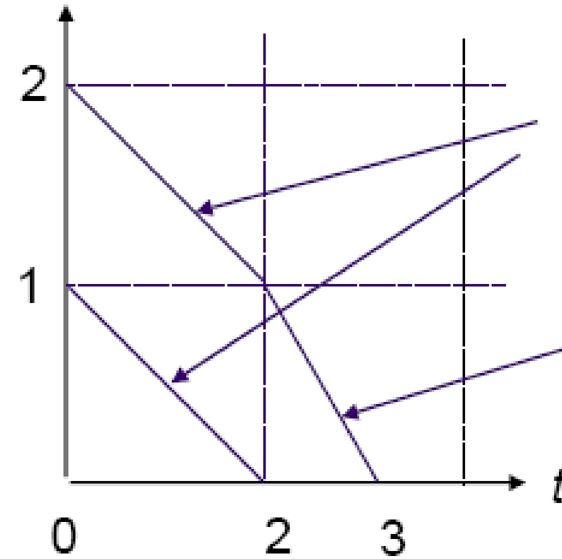
Bit-by-Bit Vs. Packet-by-Packet…

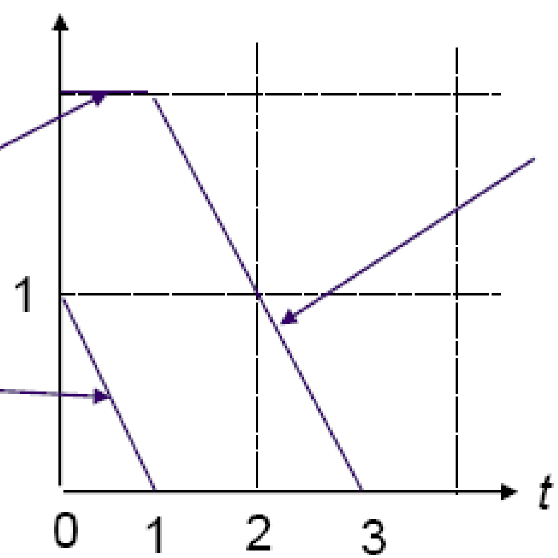
Buffer 1
at t=0

Buffer 2
at t=0



(Unequal sized packets
in the buffers)





Packet-by Packet Fair Queuing

- Let $R(t)$ denote the number of rounds made in the round-robin service discipline up to time t
- A packet of size P whose first bit is serviced at t_0 will have its last bit serviced after P rounds
  - at each round one bit of the packet is serviced

$$R(t) = R(t_0) + P$$

  - when there are more active flows the time per round will be longer than with fewer flows

Packet-by Packet Fair Queuing

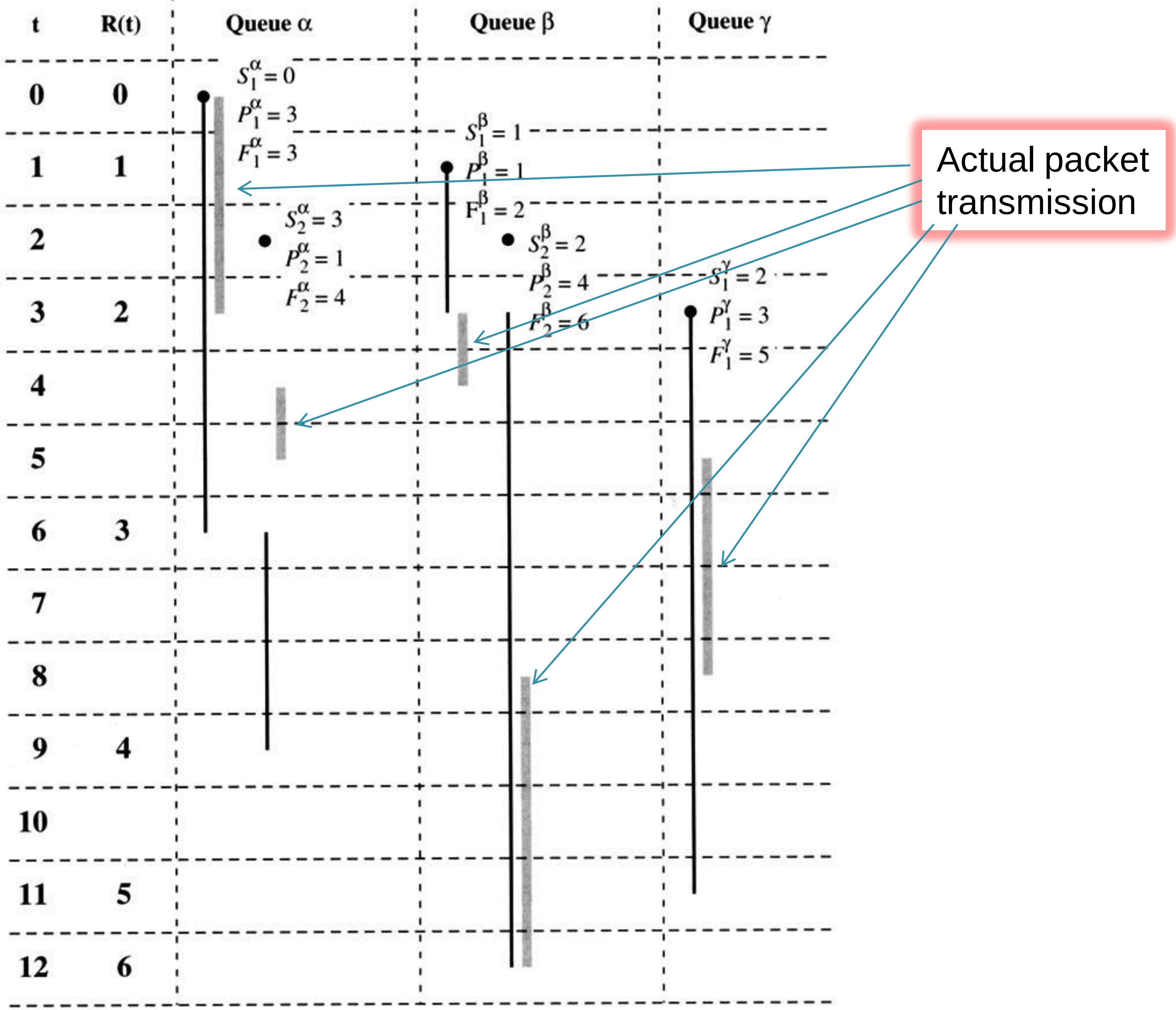
- Let t_i^α be the time packet i belonging to flow α arrives at the router
- Let S_i^α be the starting time of the packet
- Let F_i^α be the finishing time of the packet
- Let P_i^α be the packet length
- Following relations hold: $F_i^\alpha = S_i^\alpha + P_i^\alpha$

$$S_i^\alpha = \max(F_{i-1}^\alpha + R(t_i^\alpha))$$

For weighted fair queuing, use P_i^α / ϕ_α

Packet-by Packet Fair Queuing

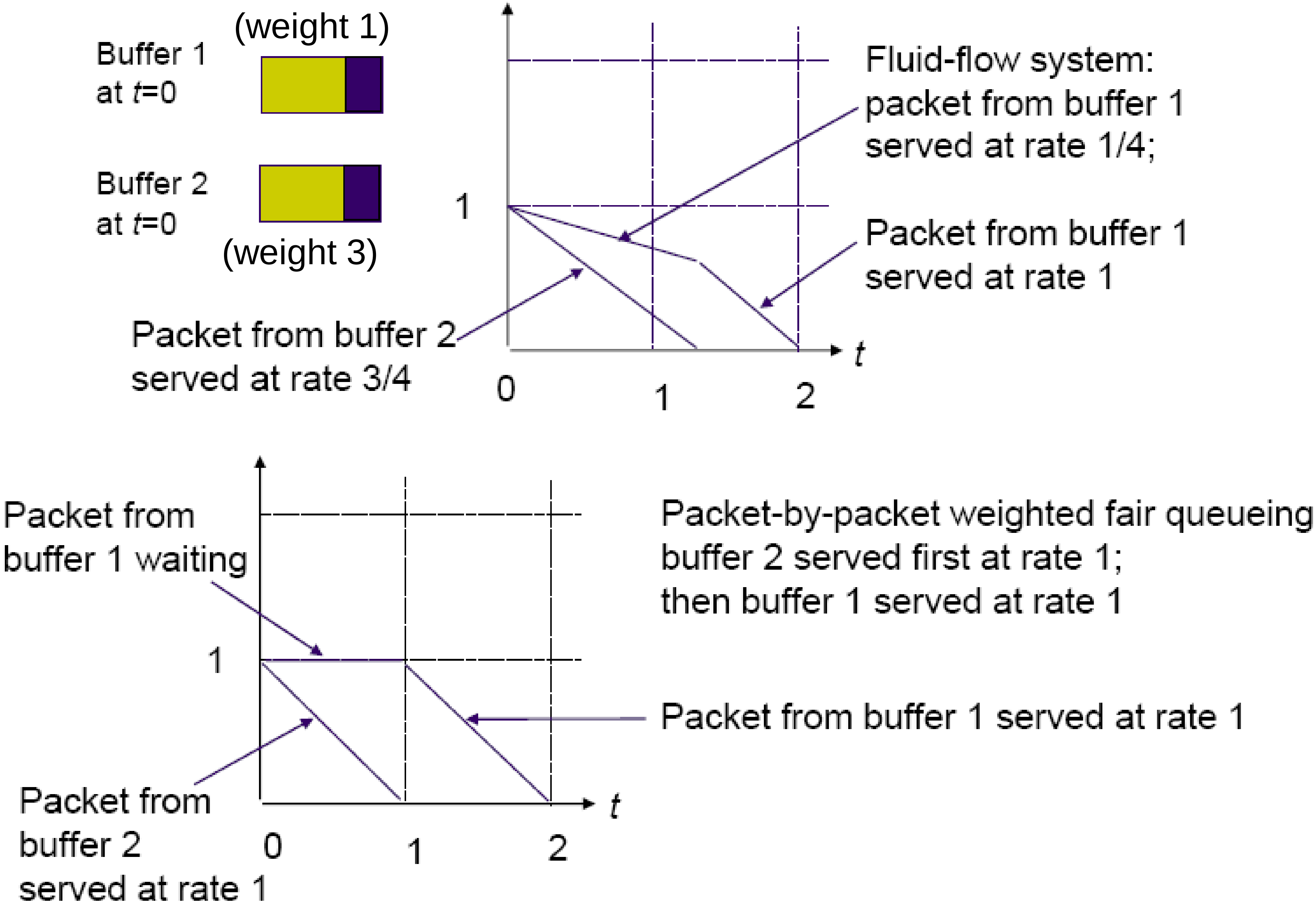
- For packet-by-packet approximation:
  - use F_i^α in defining the sending order
  - whenever a packet is finished sending, the next packet for transmission should be with the smallest F_i^α
- Preemptive version:
  - newly arriving packets with less F_i^α can preempt and ongoing packet transmission -- difficult to analyze analytically



t
R(t)
Queue α
Queue β
Queue γ
0 0
1 1
2
3 2
4
5
6 3
7
8
9 4
10
11 5
12 6
$S_1^\alpha = 0$
$P_1^\alpha = 3$
$F_1^\alpha = 3$
$S_2^\alpha = 3$
$P_2^\alpha = 1$
$F_2^\alpha = 4$
$S_1^\beta = 1$
$P_1^\beta = 1$
$F_1^\beta = 2$
$S_2^\beta = 2$
$P_2^\beta = 4$
$F_2^\beta = 6$
$S_1^\gamma = 2$
$P_1^\gamma = 3$
$F_1^\gamma = 5$
Actual packet transmission

Weighted Fair Queueing

- Addresses the reality that different users have different QoS requirements.
- Weight the queues differently. Some queues have more weight and others less.


(weight 1)
Buffer 1
at t=0
Buffer 2
at t=0
(weight 3)
1
Fluid-flow system:
packet from buffer 1
served at rate 1/4;
Packet from buffer 1
served at rate 1
Packet from buffer 2
served at rate 3/4
0
1
2
t
Packet from
buffer 1 waiting
1
Packet-by-packet weighted fair queueing:
buffer 2 served first at rate 1;
then buffer 1 served at rate 1
Packet from buffer 1 served at rate 1
Packet from
buffer 2
served at rate 1
0
1
2
t

Buffer mgmt. and Packet Drop

- Although FQ provides separate buffers, it is rarely implemented at core routers.
- With FIFO/FCFS, we need buffer management:
  - Tail drop
  - Drop on full
  - Random early drop (RED)

Packet Drop Policies

- Tail Drop
  - Sets a maximum queue length
  - Drops all incoming packets after the queue length has reached maximum
  - Is simple but has two major drawbacks: (a) allows a single flow to monopolize and (b) allows queues to build up to the maximum size and create prolonged lower link utilization

Packet Drop Policies…

- Drop on Full:
  - Can be either random drop on full or drop front on full.
  - Both solve the monopolization problem
  - Does not solve the queue becoming full problem.
  - Random early detection (RED) was proposed to address this problem.

Random Early Detection (RED)

- When there is congestion, buffers fill up and routers begin to drop packets
- TCP traffic -- goes into slow start -- reduces the network traffic -- relieves congestion
- Problems:
  - lost packets should be retransmitted
  - additional load and significant delays
  - global synchronization: several TCP flows are affected by congestion and go into slow start at the same time

RED

  - dramatic drop in network traffic -- network may be underutilized
  - TCP flows will come out of the slow start at about the same time -- another burst of traffic -- this could cause another cycle
- Solution(s):
  - bigger buffers -- not desirable
  - predict congestion and inform one TCP flow at a time to slow down

RED

- Design goals of RED:
  - congestion avoidance:
    - RED is designed to avoid congestion not to react to it
    - must predict the onset of congestion and maintain network in the efficient region of the power curve
  - global synchronization avoidance:
    - when onset of congestion is detected, router must decide which flows to notify to backoff
    - notification are implicit (dropping packets)

RED

- avoidance of bias against bursty traffic:
  - congestion is likely to occur with the arrival of bursty traffic from one or few sources
  - if only packets from bursty flows are selected for dropping, discard algorithm is biased against bursty sources
- bound on average queue length: RED should be able to control the average queue size

RED

- RED performs two functions when packets come in
  - compute average queue length *avg*
  - this is compared with two thresholds
    - less than lower threshold congestion is assumed to be non existent
    - greater than upper threshold congestion is serious
    - between the thresholds, might be onset of congestion – compute probability Pa. based on avg

RED

- RED algorithms can be summarized by the following steps:

```
calculate the average queue size avg
if avg < TH_min
    queue packet
else if TH_min ≤ avg < TH_max
    calculate probability P_a
    with probability P_a
        discard packet
    else with probability 1 - P_a
        queue packet
else if avg ≥ TH_max
    discard packet
```

RED

- In RED, we would like to space the discards such that a bursty source does not get overly penalized
- This is integrated into the computation of Pa.
  - compute a probability Pb that changes from 0 at min threshold to Pmax at max. threshold

$$F = \frac{avg - TH_{\min}}{TH_{\max} - TH_{\min}}$$

RED

- Above equation gives the fraction of the critical region – scaling factor

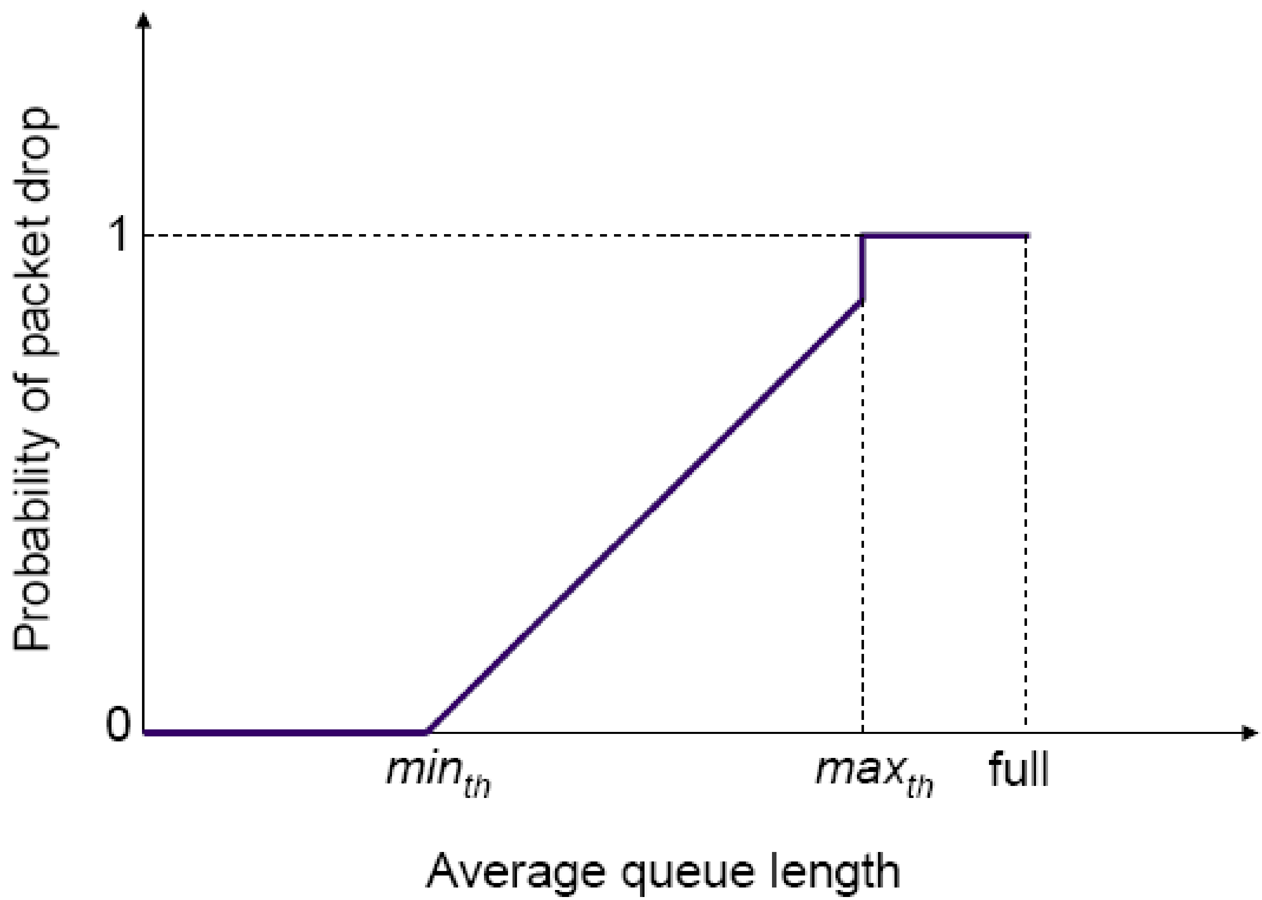
$$P_b = F * P_{max} \qquad\qquad 0 \le F \le 1$$

- Instead of using Pb directly, we compute Pa which is the probability used to discard

$$P_a = \frac{F * P_{max}}{1 - count * F * P_{max}}$$

count – number of packets sent since the last marking

RED



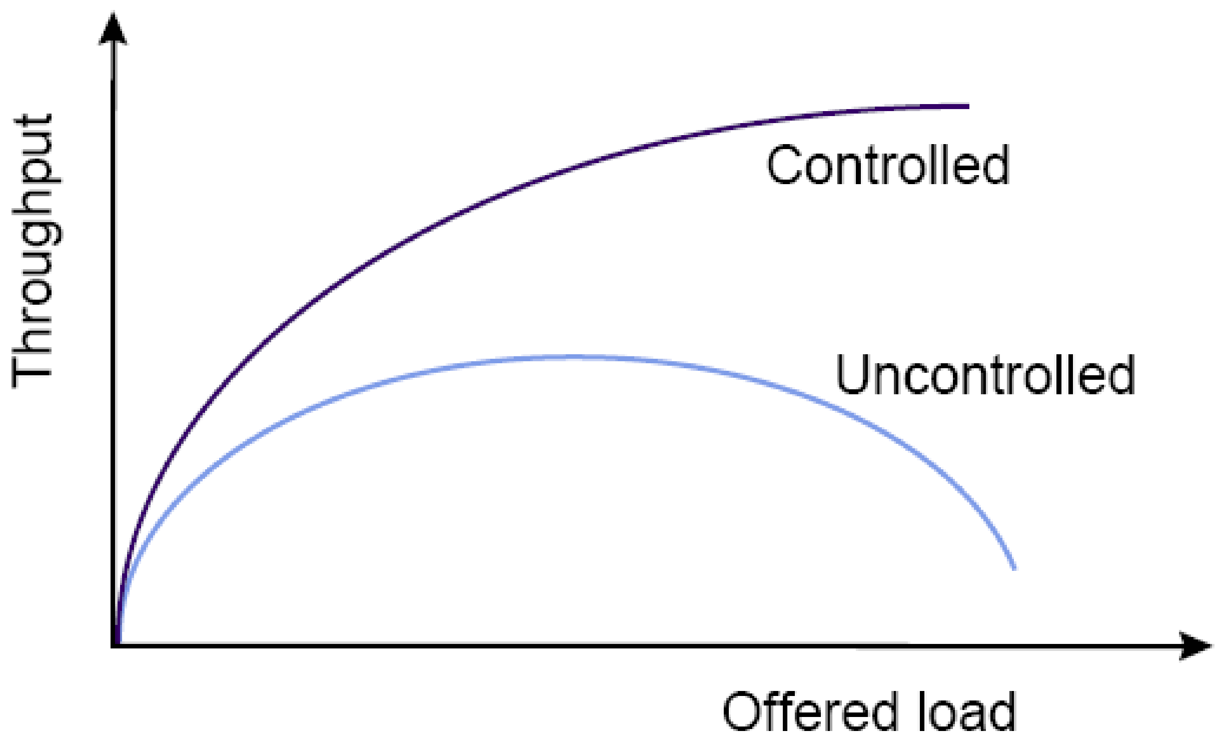
Probability of packet drop
1
0
min_{th}
max_{th}
full
Average queue length

Traffic Management at Flow Level

- At the flow level, we are concerned with managing traffic flows to ensure QoS
- Congestion control algorithms at flow level can be grouped into:
  - Open-loop control: (equivalent reservation-based approaches)
  - Closed-loop control: (equivalent to feedback based approaches)

Traffic Management at Flow Level

- Figure below shows throughput with and without congestion control. Congestion cannot be addressed by having large network buffers

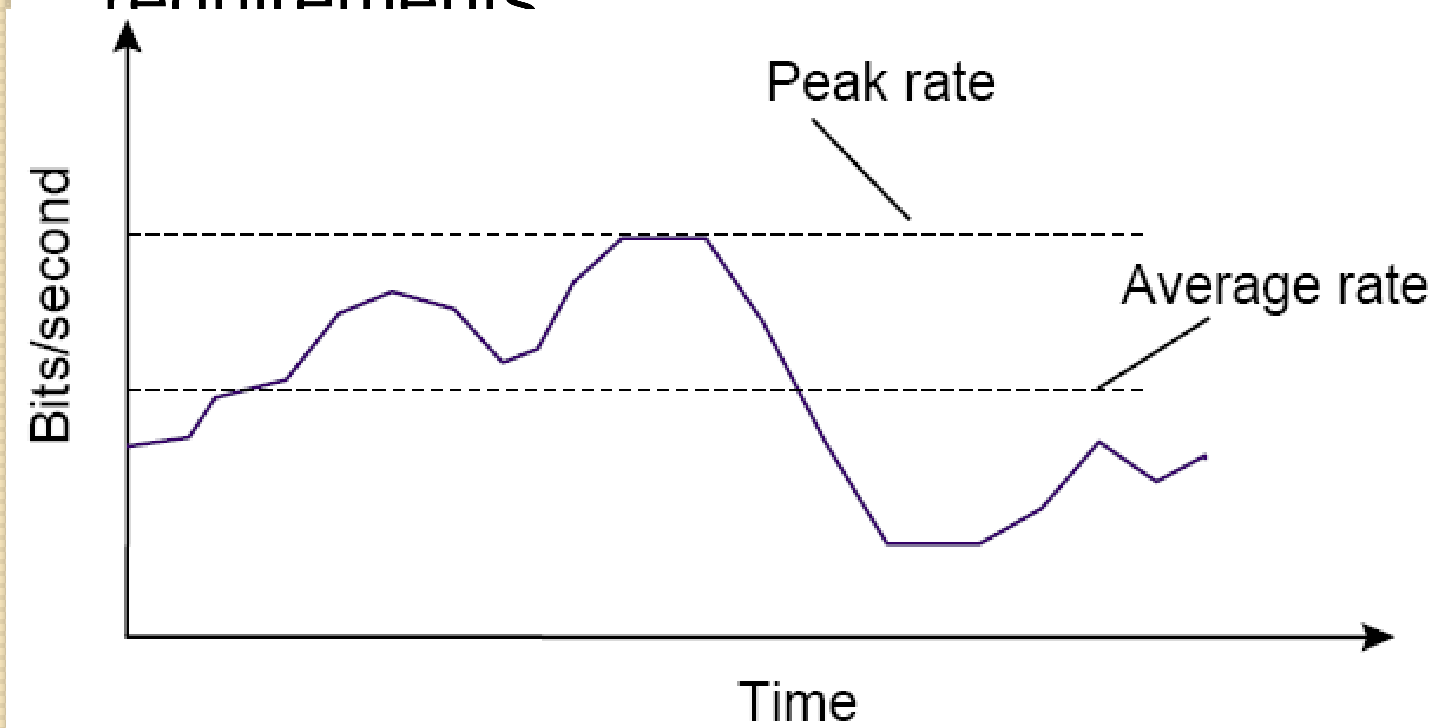


Open-Loop Traffic Control

- Open-Loop Traffic Control uses the following building blocks:
  - Admission control
  - Policing
  - Traffic Shaping

Admission Control

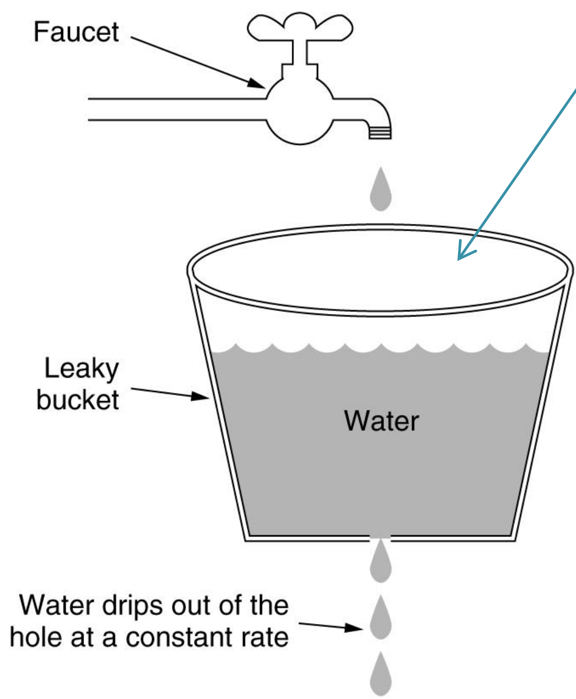
- Admission control is meant to determine whether a request for new connection should be allowed based on expected resource requirements



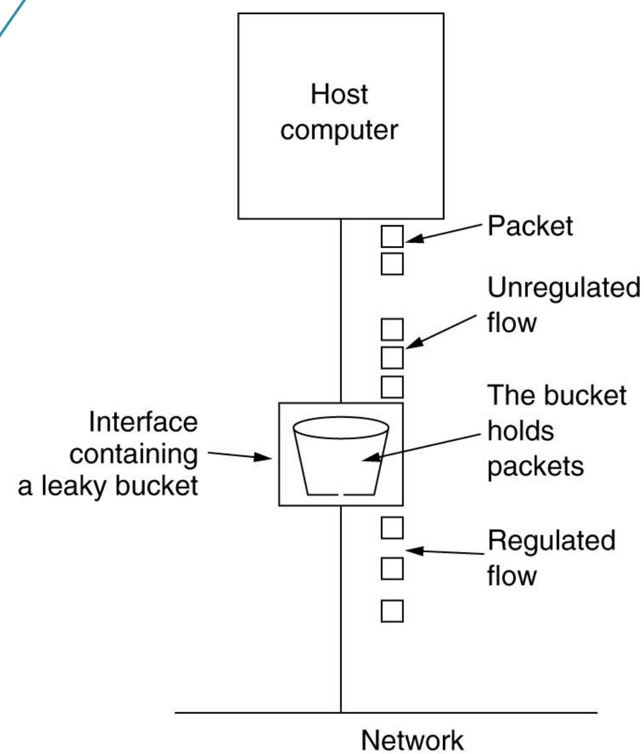
Policing

"overflowing" packets can be lost

- Policing is often implemented by a leaky bucket regulator



Leaky bucket with water

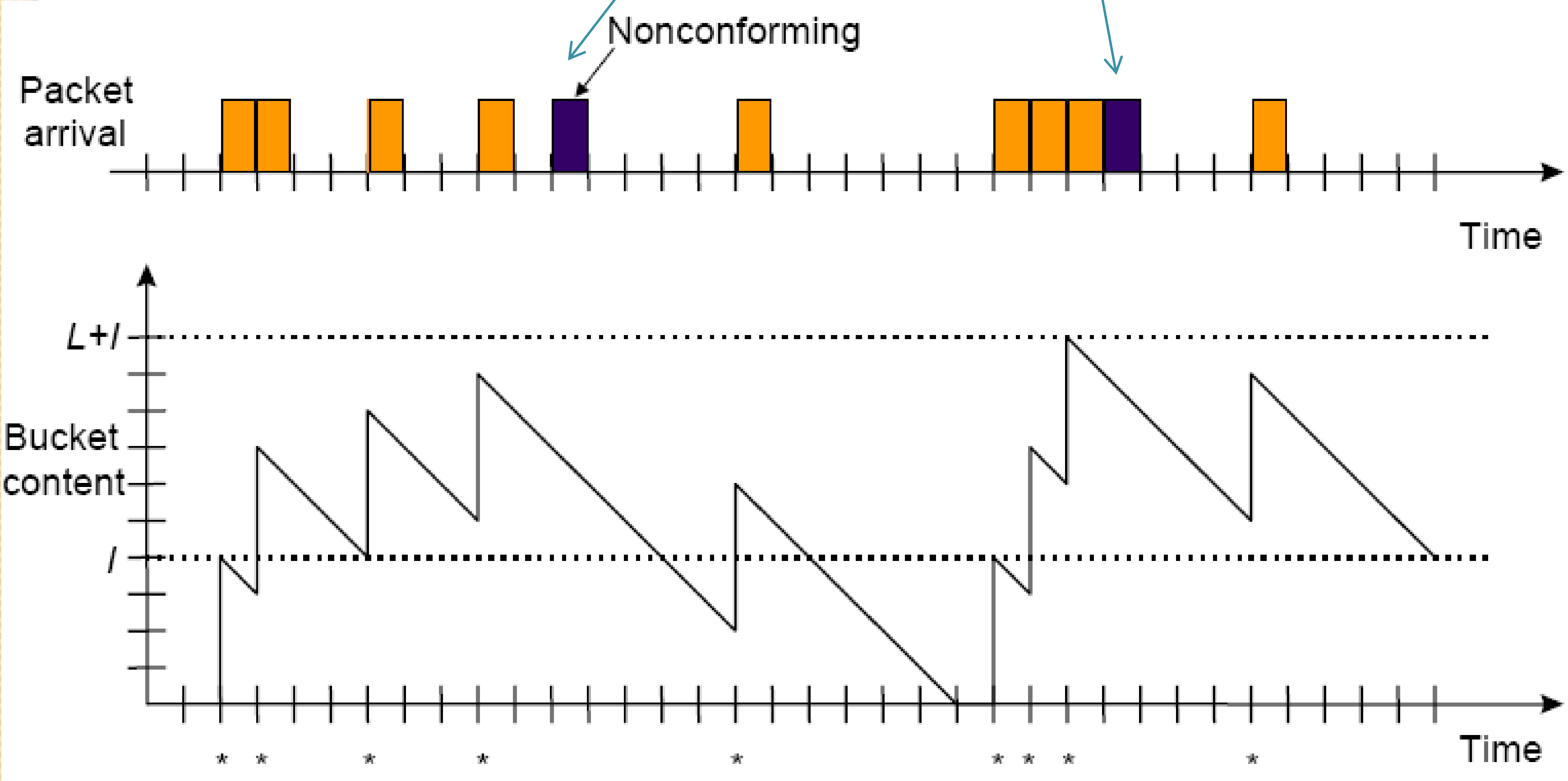


Leaky bucket with packets

Policing

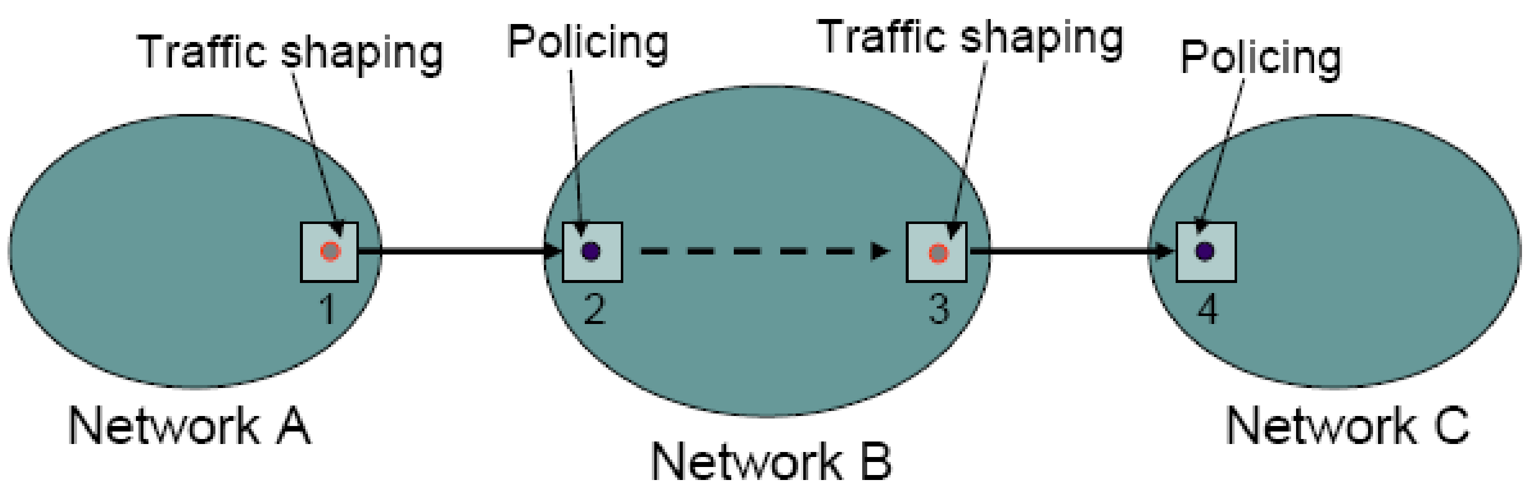
Tag or drop non-conforming packets

- Example leaky bucket policing. Counter increment 4 packet times, traffic burstiness allowed 6 packet times



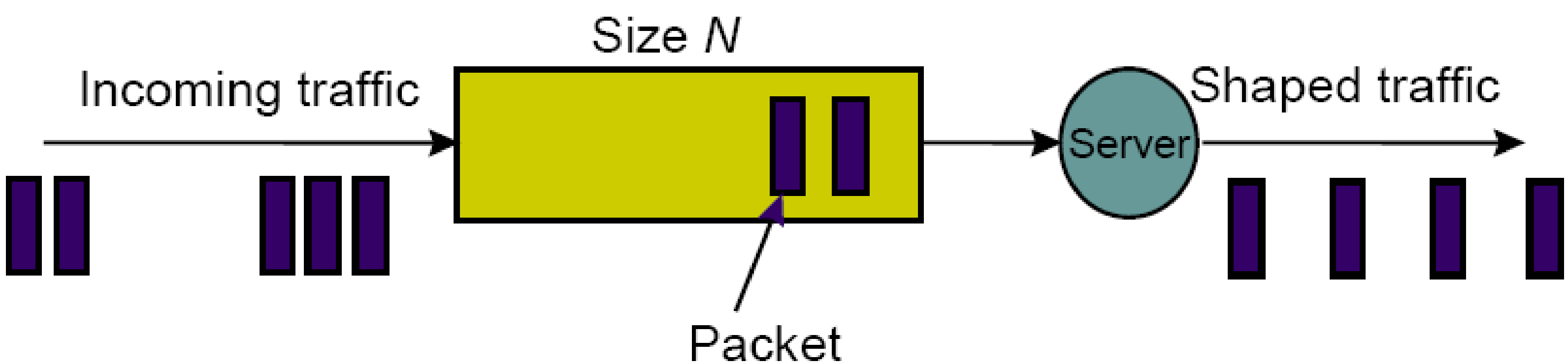
Shaping Vs. Policing

- Policing is done on incoming traffic. Shaping is done on outgoing traffic.



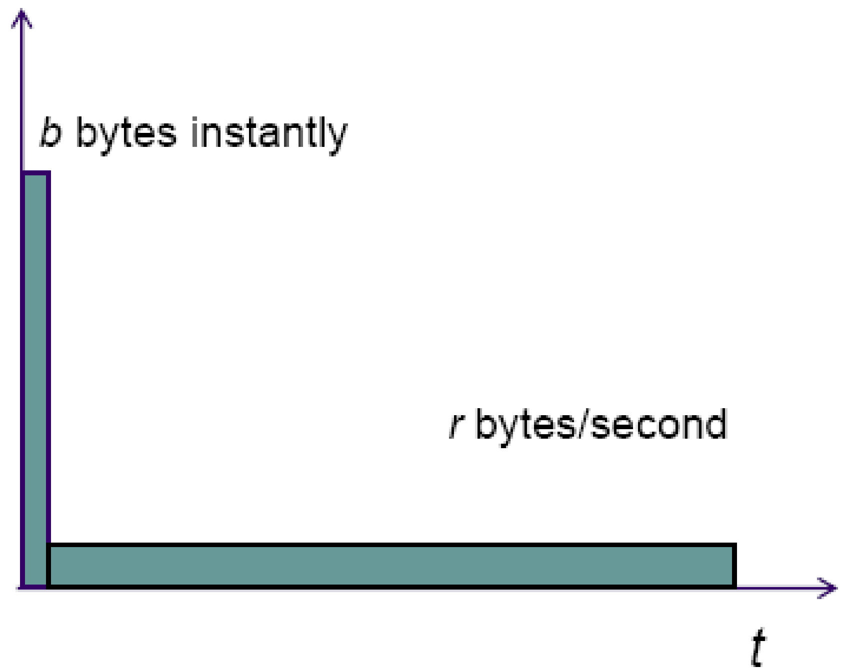
Traffic Shaping

- Traffic shaping can be done in number of ways
- Using a leaky bucket shaper.



Token Bucket Algorithm

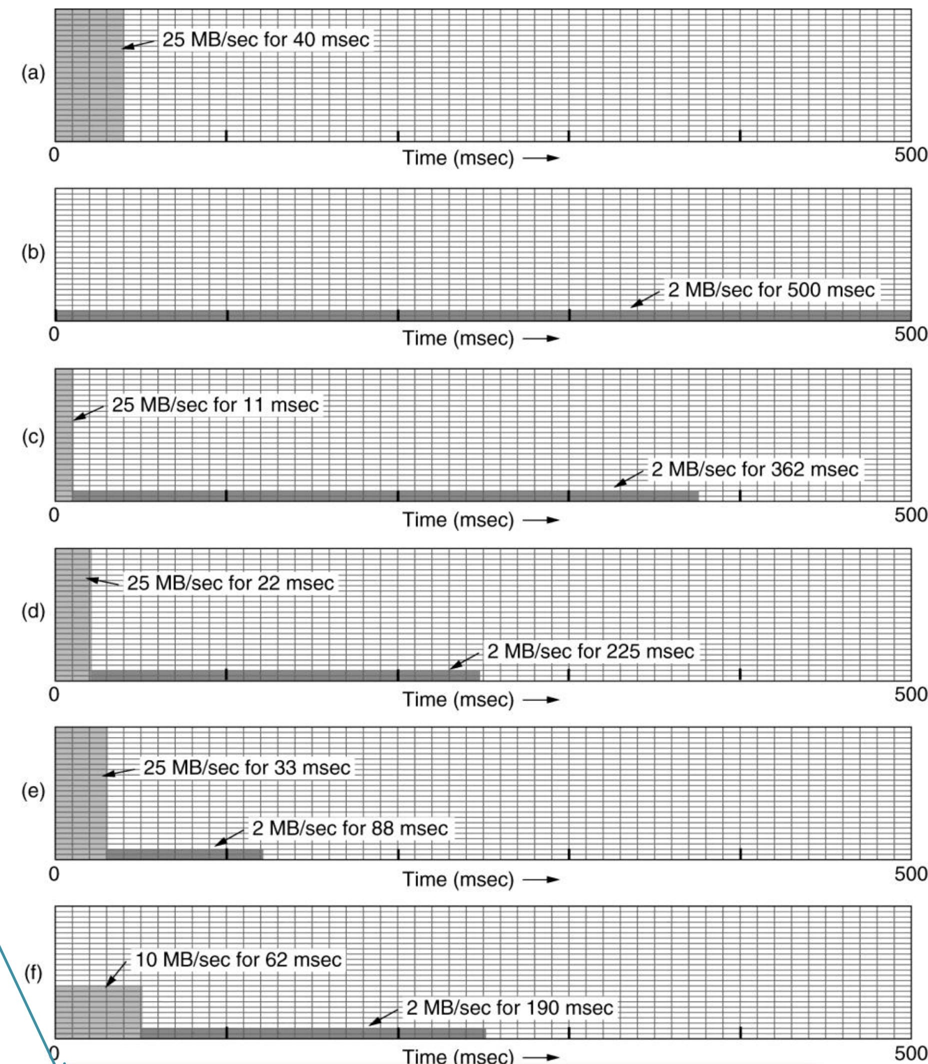
- Let b the bucket size in bytes
- Let r be the token rate in bytes/sec
- In time T, b + rT bytes can pass through



Leaky Vs. Token Bucket

Only valid for token bucket

(a) Input to a leaky bucket. (b) Output from a leaky bucket. Output from a token bucket with capacities of (c) 250 KB, (d) 500 KB, (e) 750 KB, (f) Output from a 500KB token bucket feeding a 10-MB/sec leaky bucket.



burst length S; bucket capacity b; output rate M; token arrival r;
b+Sr = MS -> S = b/(M-r)

Traffic Shaping…

- Using a token bucket shaper