# Scheduling

Bloc 7, INF 586

*Walid Dabbous*
INRIA Sophia Antipolis

# Outline

- What is scheduling
- Why we need it
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling best effort connections
- Scheduling guaranteed-service connections

# Scheduling

- Sharing resources always results in contention
  - file systems
  - long distance trunks
  - web sites
- A *scheduling discipline* resolves contention:
  - who's next?
- Key to *fairly sharing resources* and *providing performance guarantees*

# Components of a scheduling discipline

- A scheduling discipline does two things:
  - decides service order
  - manages queue of service requests
- Example:
  - consider queries awaiting web server
  - scheduling discipline decides service order
  - and also if some query should be ignored
    - storage is limited
- Allocates different service qualities to different users by its
  - choice of service order (allocate different delays)
  - choice of which request to drop (allocate different loss rate)

# Where to schedule?

- Anywhere where contention may occur
- When statistical fluctuations result in queuing
  - not in circuit switched networks
- At every layer of protocol stack
  - e.g. the web server application
- We will focus on network layer:
  - bandwidth on a specific link
  - *output* queue buffers at routers
    - assumed sufficiently fast switch fabrics

# Outline

- What is scheduling
- Why we need it
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling best effort connections
- Scheduling guaranteed-service connections

# Why do we need one?

- *Because future applications need it*
- We expect two types of future applications
  - “elastic” or best-effort (adaptive, non-real time)
    - e.g. email, some types of file transfer
  - guaranteed service (non-adaptive, real time)
    - e.g. packet voice, interactive video

# What can scheduling disciplines do?

- Give different users different qualities of service
- Scheduling disciplines can allocate
  - bandwidth
  - delay
  - loss
- Required to provide "performance guarantees"
- They also determine how *fair* the network is
  - even if best effort applications do not require performance bounds

# The Conservation Law

- FCFS is the simplest possible scheduling discipline but
  - provides no differentiation among connections
- More sophisticated scheduling discipline provides this
  - but sum of mean delays (weighted by load share) is independent from the scheduling discipline

N connections at a scheduler, $\lambda_i$ mean rate for connection *i*, $x_i$ mean service time for a packet from connection *i*. $\rho_i$ mean utilization of a link due to connection *i*, $q_i$ mean waiting time for a packet from connection i at (work-conserving) scheduler:

$$\sum_{i=1}^{N} \rho_i q_i = Cst$$

# Outline

- What is scheduling
- Why we need it
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling best effort connections
- Scheduling guaranteed-service connections

# Requirements

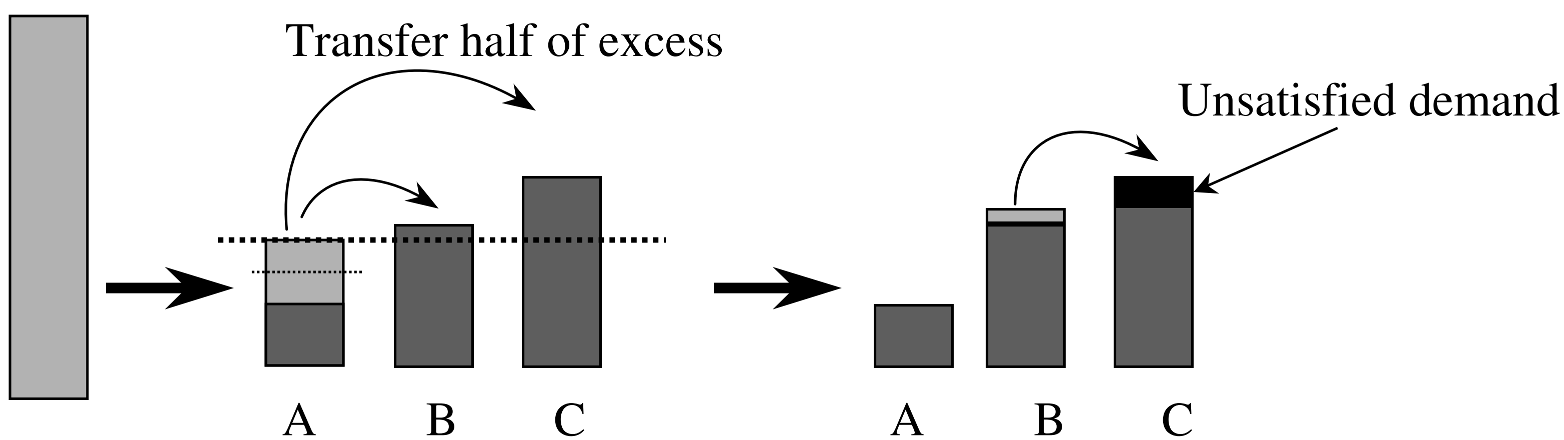
- An ideal scheduling discipline
  - is easy to implement
  - is fair (for best effort connections)
  - provides performance bounds (for GS connections)
  - allows easy *admission control* decisions (for GS)
    - to decide whether a new flow can be allowed

# Requirements: 1. Ease of implementation

- Scheduling discipline has to make a decision once every few microseconds!
- Should be implementable in a few instructions in hardware
  - for hardware: critical constraint is VLSI *space* required to maintain scheduling state and *time* to access this state
    - single shared buffer is easy
    - per-connection queuing not feasible
- Work per packet should scale less than linearly with number of active connections
  - O(N) does not scale (N=100.000 simultaneous connections in wide-area routers)

# Requirements: 2. Fairness

- Scheduling discipline *allocates* a *resource* (bw, buffers)
- An allocation is fair if it satisfies *max-min fairness*
  - maximizes the minimum share of a source whose demand is not fully satisfied
  - resources allocated in order of increasing demand
  - No source gets more than its demand
  - Sources with unsatisfied demand get an *equal* share of the resource
- Intuitively
  - each connection gets no more than what it wants
  - the excess, if any, is equally shared



# Fairness (contd.)

- Fairness is *intuitively* a good idea for best effort connection
  - GS connections should pay the network
  - for network operators fairness is not a concern
- Fairness is a *global* objective, but scheduling is local
- Each endpoint must restrict its flow to the *smallest* fair allocation
- Dynamics + delay => global fairness may never be achieved
- But it also provides *protection*
  - traffic hogs cannot overrun others
  - automatically builds "*firewalls*" around heavy users
- NB: policing at network entrance provides protection, but not fairness

# Requirements: 3. Performance bounds

- What is it?
  - A way to obtain a desired level of service
  - restricted by conservation law
    - cannot give *all* connections delay lower than FCFS
- Contract between user and network
  - user somehow communicates perf req to operator
  - hard to guarantee end to end performance bounds
- Performance bounds can be *deterministic* (holds for every packet) or *statistical* (probabilistic bound)
- Common parameters are
  - bandwidth
  - delay
  - delay-jitter
  - loss (will consider zero loss)

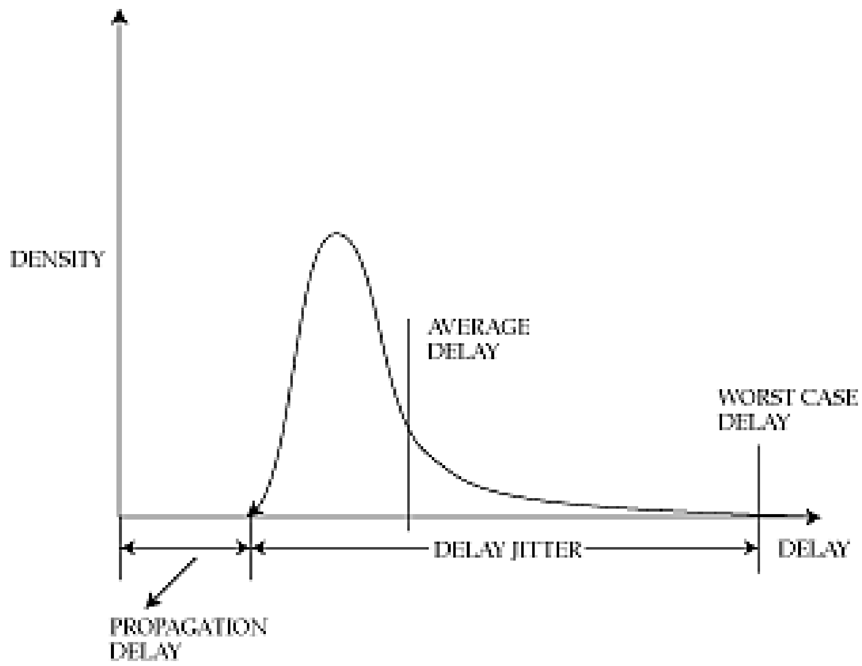
# Bandwidth

- Specified as minimum bandwidth measured over a prespecified interval
- E.g. > 5Mbps over intervals of > 1 sec
- Meaningless without an interval!
- Can be a bound on average (sustained) rate or peak rate
- Peak is measured over a ‘small’ interval
- Average is asymptote as intervals increase without bound

- Bw bound required for all GS connections

# Delay and delay-jitter

- Bound on some parameter of the delay distribution curve




- GS networks are expected to specify and guarantee only the deterministic or statistical *worst-case* delay (every other connection behaves in the worst possible manner)

# Req'ments: 4. Ease of admission control

- Admission control needed to provide QoS
- Decide given the currents connections whether to accept a new one without jeopardizing the performance of existing connections
- Overloaded resource cannot guarantee performance
  - but performance guarantees should not lead to network underutilization
- Choice of scheduling discipline affects ease of admission control algorithm

# Outline

- What is scheduling
- Why we need it
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling best effort connections
- Scheduling guaranteed-service connections

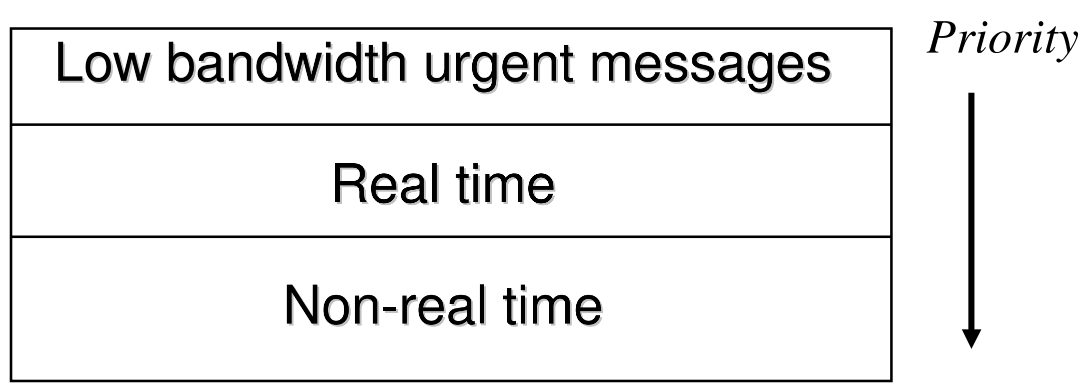
# Fundamental choices

- Degrees of freedom in designing a scheduling discipline

1. Number of priority levels
2. Work-conserving vs. non-work-conserving
3. Degree of aggregation within a level
4. Service order within a level

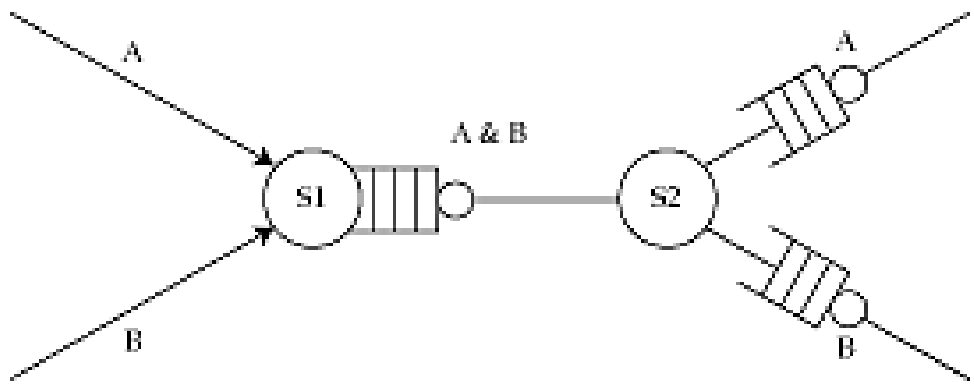
# Choices: 1. Priority scheduling

- Packet is served from a given priority level only if no packets exist at higher levels (*multilevel priority with exhaustive service*)
- Highest level gets lowest delay
- Watch out for starvation! (admission control for all but lowest priority whose ‘server’ is on ‘vacation’ when server higher priority)
- Usually map priority levels to delay classes



# Choices: 2. Work conserving vs. non-work-conserving

- Work conserving discipline is never idle when packets await service
- Why bother with non-work conserving?
- Avoid burst 'accumulation' that
  - requires larger buffers
  - results in higher jitter



# Non-work-conserving disciplines

- Key conceptual idea: delay packet till *eligible*
- Reduces delay-jitter => fewer buffers in network
- How to choose eligibility time?
  - rate-jitter regulator
    - bounds maximum outgoing rate
    - $E(1) = A(1)$; $E(k+1) = \max(E(k)+X_{min}, A(k+1))$
    - where Xmin is inverse of peak rate
  - delay-jitter regulator
    - compensates for variable delay at previous hop
    - $E(0,k) = A(0,k)$; $E(i+1,k) = E(i,k) + D + L$
    - D is max delay at previous switch, L max delay on transmission link between switch i and i+1

# Do we need non-work-conservation?

- Can remove delay-jitter at an endpoint instead
  - but also reduces size of switch buffers…
- Increases mean delay
  - not a problem for *playback* applications
- Wastes bandwidth
  - can serve best-effort packets instead
- Always punishes a misbehaving source
  - even if bandwidth is available
- Bottom line: not too bad, implementation cost may be the biggest problem (calendar queue)

# Choices: 3. Degree of aggregation

- More aggregation
  - less state
  - cheaper
    - smaller VLSI
    - less to advertise
  - BUT: less individualization
- Solution
  - aggregate to a *class*, members of class have same performance requirement
  - no protection within class
    - 'share' burst effect

# Choices: 4. Service within a priority level and an aggregation class

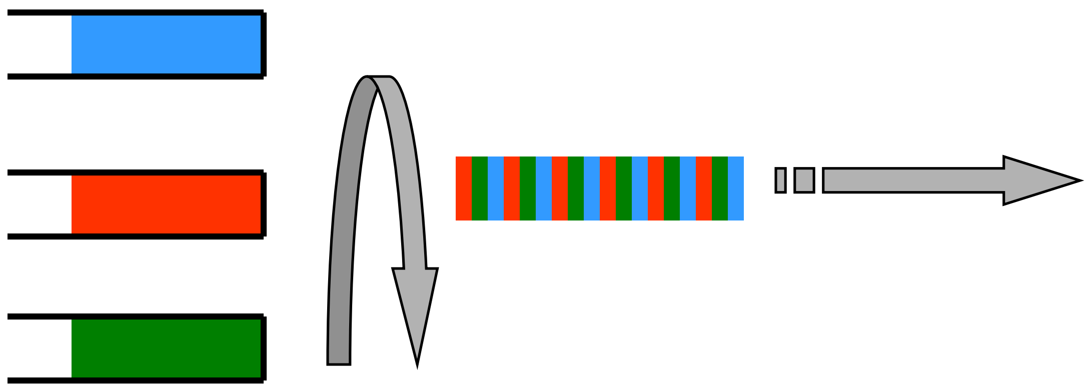
- In order of arrival (FCFS) or in order of a service tag
- Service tags => can arbitrarily reorder queue
  - Need to sort queue, which can be expensive
- FCFS
  - bandwidth hogs win (no protection)
    - greediness is rewarded
  - no guarantee on delays
- Service tags
  - with appropriate choice, both protection and delay bounds possible

# Outline

- What is scheduling
- Why we need it
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling best effort connections
- Scheduling guaranteed-service connections

# Scheduling best-effort connections

- Main requirement is (max-min) *fairness*
- Achievable using *Generalized processor sharing (GPS)*
  - Visit each non-empty (virtual) queue in turn
  - Serve infinitesimal from each
    - in any finite time interval it can visit every logical queue at least one
  - achieves max-min fairness by definition
  - may serve data in proportion to given *weight*



# More on GPS

- GPS is unimplementable!
  - we cannot serve infinitesimals, only packets
- No packet discipline can be as fair as GPS
  - while a packet is being served, we are unfair to others
- Degree of unfairness can be bounded
- **Define**: *work(i,a,b)* = # bits transmitted for connection i in time [a,b]
- *Absolute* fairness bound (AFB) for discipline S
  - Max (work_GPS(i,a,b) - work_S(i, a,b))
- *Relative* fairness bound (RFB) for discipline S
  - Max (work_S(i,a,b) - work_S(j,a,b))

# What next?

- We can’t implement GPS
- So, lets see how to emulate it
- We want to be as fair as possible
- But also have an efficient implementation

# Weighted round robin

- RR: Serve a packet from each non-empty queue in turn
- Unfair if packets are of different length or weights are not equal
- Different weights, fixed packet size
  - serve more than one packet per visit, after normalizing to obtain integer weights
- Different weights, variable size packets
  - normalize weights by mean packet size
    - e.g. weights {0.5, 0.75, 1.0}, mean packet sizes {50, 500, 1500}
    - normalize weights: {0.5/50, 0.75/500, 1.0/1500} = { 0.01, 0.0015, 0.000666}, normalize again {60, 9, 4}

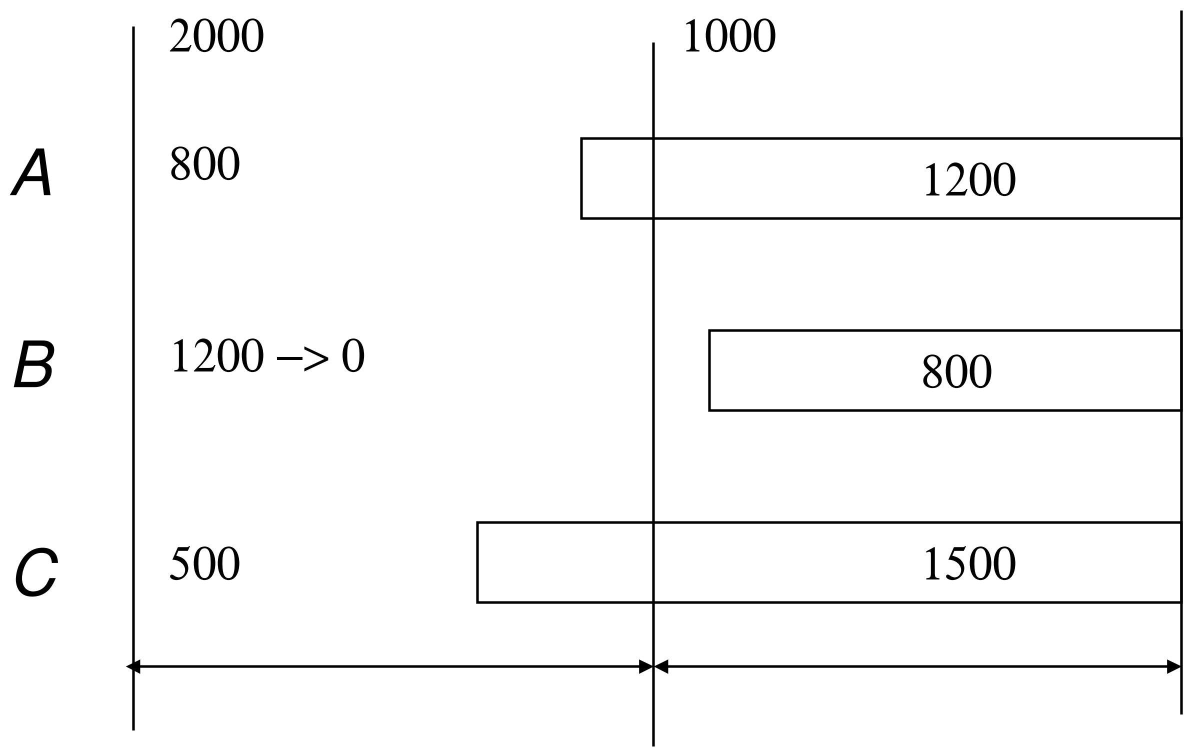
# Problems with Weighted Round Robin

- With variable size packets and different weights, need to know mean packet size in advance
  - what about compressed video?
- Fair on time scales longer than a round time
  - Can be unfair for long periods of time
    - if a connection has a small weight
    - or number of connections is large
- E.g.
  - T3 trunk with 500 connections, each connection has mean packet length 500 bytes, 250 with weight 1, 250 with weight 10
  - Each packet takes 500 * 8/45 Mbps = 88.8 microseconds
  - Round time =2750 * 88.8 = 244.2 ms

# Deficit round-robin

- Modifies WRR to handle variable packet sizes
  - without knowing mean packet size of each connection in advance
- Initialize ‘deficit counter’ to zero
- visit each queue
  - if (DC + quantum) $\geq$ size of packet at head of queue -> serve
    - and decrement deficit counter
- Easy to implement
- But fair on large time scale

# Example



# Weighted Fair Queueing (WFQ)

- Deals better with variable size packets and weights
  - like DRR
- GPS is fairest discipline
- Find the *finish time* of a packet, *had we been doing GPS*
- Then serve packets in order of their finish times

# WFQ

- Suppose, in each *round,* the server served one bit from each active connection
- *Round number* is the number of rounds already completed
  - can be fractional
- If a packet of length $p$ arrives to an empty queue when the round number is $R$, it will complete service when the round number is $R + p$ => *finish number* is $R + p$
  - independent of the number of other connections!
- If a packet arrives to a non-empty queue, and the previous packet has a finish number of $f$, then the packet's finish number is $f+p$
- Serve packets in order of finish numbers

# Evaluation

- Pros
  - like GPS, it provides protection
  - can obtain worst-case end-to-end delay bound
  - gives users incentive to use intelligent congestion control (and also provides rate information implicitly)
- Cons
  - needs per-connection state
  - implementation complexity
  - explicit sorting of output queue

# Outline

- What is scheduling
- Why we need it
- Requirements of a scheduling discipline
- Fundamental choices
- Scheduling best effort connections
- Scheduling guaranteed-service connections

# Scheduling guaranteed-service connections

- With best-effort connections, goal is fairness
- With guaranteed-service connections
  - what performance guarantees are achievable?
  - how easy is admission control?
- We now study some scheduling disciplines that provide performance guarantees

# WFQ

- Turns out that WFQ also provides performance guarantees
- Bandwidth bound
  - ratio of weights * link capacity
  - e.g. connections with weights 1, 2, 7; link capacity 10
  - connections get at least 1, 2, 7 units of b/w each
- End-to-end delay bound
  - assumes that the connection doesn't send 'too much' (otherwise its packets will be stuck in queues)
  - more precisely, connection should be *leaky-bucket* regulated
  - # bits sent in time $[t_1, t_2] <= \rho\ (t_2 - t_1) + \sigma$

# Parekh-Gallager theorem

- Let a connection be allocated weights at each WFQ scheduler along its path, so that the least bandwidth it is allocated is *g*
- Let it be leaky-bucket regulated such that # bits sent in time $[t_1, t_2] <= \rho (t_2 - t_1) + \sigma$
- Let the connection pass through *K* schedulers, where the *k*th scheduler has a link rate *r(k)*
- Let the largest packet allowed in the network be *Pmax*
- The transmission delay is bounded by:

$$\boxed{D^*(i) \leq \sigma(i)/g(i) + \sum_{k=1}^{K-1} P\max(i)/g(i,k) + \sum_{k=1}^{K} P\max/r(k)}$$

# Significance

- Theorem shows that WFQ can provide worst-case end-to-end delay bounds
- So WFQ provides both fairness and performance guarantees
- Bound holds regardless of cross traffic behavior
- Can be generalized for networks where schedulers are variants of WFQ, and the link service rate changes over time
- But bounds are VERY large and useless

# Problems

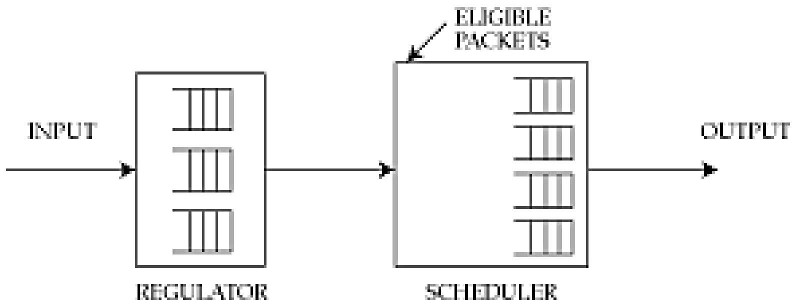
- To get a delay bound, need to pick *g*
  - the lower the delay bounds, the larger *g* needs to be
  - large *g* => exclusion of more competitors from link
  - *g* can be very large, in some cases 80 times the peak rate!
- Sources must be leaky-bucket regulated
  - but choosing leaky-bucket parameters is problematic
- WFQ couples delay and bandwidth allocations
  - low delay requires allocating more bandwidth
  - wastes bandwidth for low-bandwidth low-delay sources

# Delay-Earliest Due Date

- Earliest-due-date: packet with earliest deadline selected
- Delay-EDD prescribes how to assign deadlines to packets
- A source is required to send slower than its *peak rate*
- Bandwidth at scheduler reserved at peak rate
- admission control ensures that delay bound will be met
- Deadline = ‘expected’ arrival time + delay bound (time at which it should be sent, had it been received according to the contract)
  - If a source sends faster than contract, delay bound will not apply
- Each packet gets a hard delay bound
- E2E Delay bound is *independent* of bandwidth requirement
  - but reservation is at a connection’s peak rate
- Implementation requires per-connection state and a priority queue

# Rate-controlled scheduling

- A *class* of disciplines
  - two components: regulator and scheduler
  - incoming packets are placed in regulator where they wait to become *eligible*
  - then they are put in the scheduler
- Regulator *shapes* the traffic, scheduler provides performance guarantees

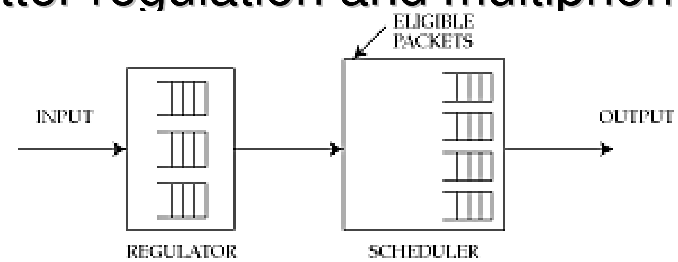


# Analysis

- First regulator on path monitors and regulates traffic => bandwidth bound
- End-to-end delay bound
  - delay-jitter regulator
    - reconstructs traffic => end-to-end delay is fixed (= worst-case delay at each hop)
  - rate-jitter regulator
    - partially reconstructs traffic
    - can show that end-to-end delay bound is smaller than (sum of delay bound at each hop + delay at first hop)

# Decoupling

- Can give a low-bandwidth connection a low delay without overbooking
- E.g consider connection A with rate 64 Kbps sent to a router with rate-jitter regulation and multipriority FCFS scheduling




- After sending a packet of length *P*, next packet is eligible at time (now + *P*/64 Kbps)
- If placed at highest-priority queue, all packets from A get low delay
- Can decouple delay and bandwidth bounds, unlike WFQ

# Evaluation

- Pros
  - flexibility: ability to emulate other disciplines
  - can decouple bandwidth and delay assignments
  - end-to-end delay bounds are easily computed
  - do not require complicated schedulers to guarantee protection
  - can provide delay-jitter bounds
- Cons
  - require an additional regulator at each output port
  - delay-jitter bounds at the expense of increasing mean delay
  - delay-jitter regulation is expensive (clock synch, timestamps)

# Summary

- Two sorts of applications: best effort and guaranteed service
- Best effort connections require fair service
  - provided by GPS, which is unimplementable
  - emulated by WFQ and its variants
- Guaranteed service connections require performance guarantees
  - provided by WFQ, but this is expensive
  - may be better to use rate-controlled schedulers