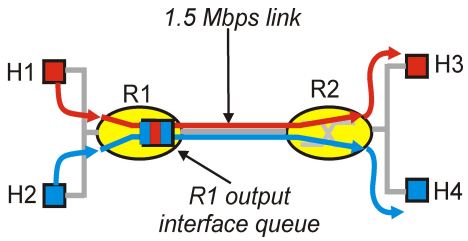
Provisioning QOS in IP Networks

Thus far: "making the best of 'best effort' delivery"

Goal: next generation Internet with QoS guarantees

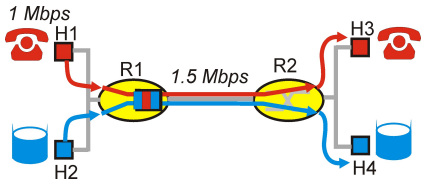
- Integrated Services: firm guarantees
- RSVP: signaling for resource reservations
- Differentiated Services: differential guarantees

- A simple model for sharing and congestion studies:



Principles for QOS Guarantees

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

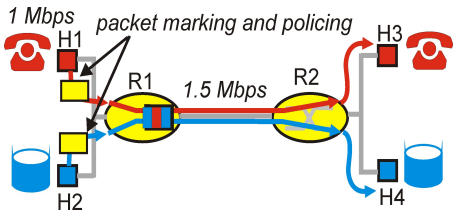


Principle 1

Packet marking needed for router to distinguish between different classes, and a new router policy to treat packets accordingly

Principles for QOS Guarantees

- What if applications misbehave (audio sends higher than declared rate)
  - policing: enforce source adherence to bandwidth allocations
- Marking and policing at network edge:

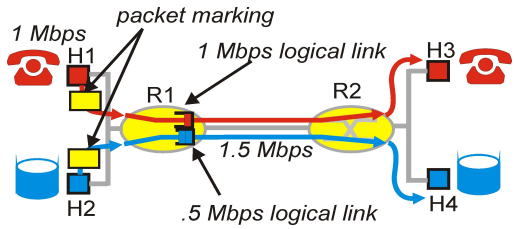


Principle 2

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

Principles for QOS Guarantees

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

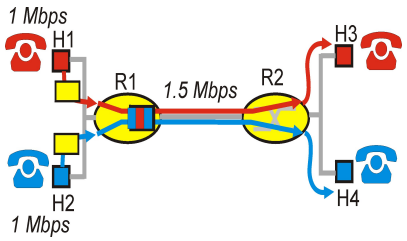


Principle 3

While providing isolation, it is desirable to use resources as efficiently as possible (*work-preserving*)

Principles for QOS Guarantees

- Basic fact of life: cannot support traffic demands beyond link capacity



Principle 4

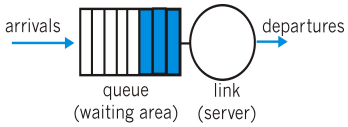
Call Admission: flow declares its needs, network may block call (e.g., busy signal) if it cannot meet flow's needs

Summary of QoS Principles

- Packet Classification
  - Associate each packet with a "reservation" at the routers
- Isolation: Scheduling and policing
  - Manage queues so that packets receive the requested service
- High resource utilization
  - Efficient use of resources in the network (work-preserving)
- Call admission
  - Decide if new flow can be supported without sacrificing the QoS levels of the existing flows
- Let's next look at mechanisms for achieving these principles.

Scheduling Mechanisms

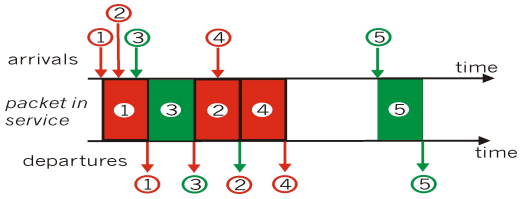
- Scheduling: choose next packet to send on link
- FIFO (first in first out) scheduling: send in order of arrival to queue
  - discard policy: if packet arrives to full queue: who to discard?
    - Tail drop: drop arriving packet
    - Priority: drop/remove on priority basis
    - Random: drop/remove randomly



Scheduling Policies

Round robin scheduling:

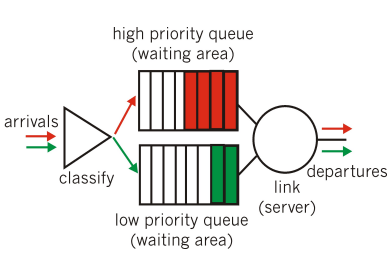
- multiple classes
- cyclically scan class queues, serving one from each class (if available)

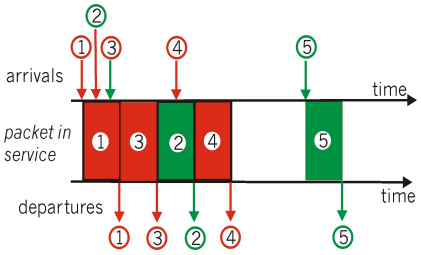


Scheduling Policies

Priority scheduling: transmit highest priority queued packet first

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



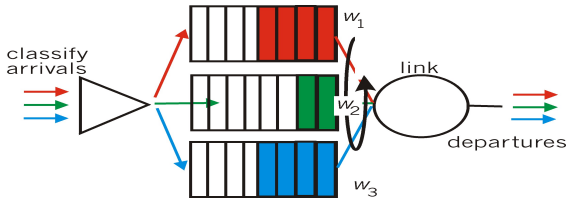


Non-preemptive service

Scheduling Policies

Weighted Fair Queuing:

- generalized Round Robin
- each class gets weighted amount of service in each cycle



Queuing Disciplines

- Each router must implement some queuing discipline
- Queuing allocates both bandwidth and buffer space:
  - Bandwidth: (serving discipline) which packet to serve (transmit) next; e.g., *FCFS*
  - Buffer space: (dropping scheme) which packet to drop next (when required); e.g., *tail drop*
- Queuing also affects latency

Typical Internet Queuing

- FIFO + drop-tail
  - Simplest choice
  - Used widely in the Internet
  - FIFO (first-in-first-out)
    - Implies single class of traffic
  - Drop-tail
    - Arriving packets get dropped when queue is full regardless of flow or importance
- Important distinction:
  - FIFO: scheduling discipline
  - Drop-tail: drop policy

FIFO + Drop-tail Problems

- Leaves responsibility of congestion control to edges (e.g., TCP)
- Does not separate between different flows
- No policing: if sending more packets → getting more service
- Synchronization: end hosts react to the same events

Active Queue Management & Design

- Design active router queue management to aid congestion control
- Why?
  - Routers can distinguish between propagation delay and persistent queuing delay (while end points cannot)
  - Routers can decide on transient congestion, based on workload
- Modify both routers and hosts
  - Explicit congestion notification info in packet header
    - E.g. DECbit – not used in TCP/IP
    - E.g. ECN – explicit congestion notification to work w/TCP
- Modify routers (while hosts use TCP)
  - Fair queuing
    - Per-connection buffer allocation
  - RED (Random Early Detection)
    - Drop packet or set bit in packet header as soon as congestion starts

What is Fairness and Fairness Goals

- At what granularity?
  - Flows, connections, domains?
- What if flows/connections have different RTTs/links/etc.
  - Should they share a link fairly or be TCP-fair?
- A tough question to answer – typically design mechanisms instead of policy
- Is TCP CC (TCP Reno) fair ?
- Allocate resources fairly
- Isolate ill-behaved users
  - Router does not send explicit feedback to source
  - Still needs E2E congestion control
- Still achieve statistical multiplexing
- One flow can fill the entire pipe if no contenders
  - Work conserving → scheduler never idles link if there's a packet to send

Max-min Fairness

- Allocate user with "small" demand of what it wants, evenly divide unused resources to "big" users
- Formally:
  - Resources allocated in terms of increasing demand
  - No source gets resource share larger than its demand
  - Sources with unsatisfied demands get equal share of resource
- Assume sources S_1,..,S_n, with resource demands X_1, …, X_n in ascending order ($X_1 \leq X_2 \leq ... \leq X_n$.
- Assume channel capacity C.
  - If $X_1 \geq C/n \rightarrow$ give C/n to S_1
  - If $X_1 < C/n \rightarrow$ give X_1 to S_1 and divide the excess ($C/n - X_1$) to other sources, with each to receive $C/n + (C/n - X_1)/(n-1)$
  - *If* $X_2 \geq C/n + (C/n - X_1)/(n-1)$ give this to S_2
  - Otherwise, give X_2 to S_2 and divide the excess $(C/n + (C/n - X_1)/(n-1) - X_2)$ to the remaining $n-2$ sources; i.e.,
    $$C/n + (C/n - X_1)/(n-1) + (C/n + (C/n - X_1)/(n-1) - X_2)/(n-2)$$
  - Repeat for other sources

Implementing max-min Fairness

- At the end, each source gets: (1) no more than what it asks for, and, (2) if its demand was not satisfied, no more than what any other source with a higher index got.
- The scheme is called a *max-min fair* allocation, since the scheme maximizes the minimum share of a source whose demand is not fully satisfied.
- Bitwise round robin among all queues
- Why not simple round robin?
  - Variable packet length → can get more service by sending bigger packets
  - Unfair instantaneous service rate
    - What if arrive just before/after packet departs?
  - Although fair in a long term

Bit-by-bit Round Robin

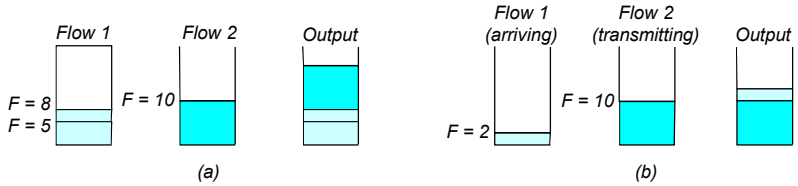
- Single flow:
  a logical clock ticks when a bit is transmitted
- For packet *i:*
  - P_i = time (duration) to transmit packet *i*
  - A_i = arrival time
  - S_i = begin transmit time
  - F_i = finish transmit time

$$F_i = S_i + P_i = max\ (F_{i-1}, A_i) + P_i$$

- Multiple flows:
  clock ticks when a bit from all active flows is transmitted → round number
- Mapping bit-by-bit schedule onto packet transmission schedule
- Transmit packet with the lowest F_i at any given time
- Fair queuing simulates bit-by-bit RR

FQ Algorithm

- For multiple flows
  - calculate F_i for each packet that arrives on each flow
  - treat all F_i's as timestamps
  - next packet to transmit is one with lowest timestamp
- Not perfect: can't preempt current packet
- Example



Delay Allocation

- Reduce delay for flows that are using less than fair share
  - Pull back arrival times for sources whose queues drain temporarily
- Schedule based on B_i instead of F_i
  - $F_i = P_i + max\ (F_{i-1}, A_i) \rightarrow B_i = P_i + max\ (F_{i-1}, A_i - \delta)$
  - If $A_i < F_{i-1}$, flow is active and δ has no effect
  - If $A_i > F_{i-1}$, flow is inactive and δ determines how much history to take into account
    - Infrequent senders do better when history is used

Fair Queuing Tradeoffs

- FQ can help control congestion by monitoring flows
  - Non-adaptive flows can still be a problem – why?
- Complex state
  - Must keep queue per flow
    - Hard in routers with many flows (e.g., backbone routers)
    - Flow aggregation is a possibility (e.g., do fairness per domain)
- Complex computation
  - Classification into flows may be hard
  - Must keep queues sorted by finish times
  - The round duration changes whenever the flow count changes