15-441 Computer Networking

Lecture 22 – Queue Management and QoS

Overview

- Queue management & RED
- Fair-queuing
- Why QOS?
- Integrated services

Queuing Disciplines

- Each router must implement some queuing discipline
- Queuing allocates both bandwidth and buffer space:
  - Bandwidth: which packet to serve (transmit) next
  - Buffer space: which packet to drop next (when required)
- 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 completely to the edges (e.g., TCP)
- Does not separate between different flows
- No policing: send more packets → get more service
- Synchronization: end hosts react to same events

FIFO + Drop-tail Problems

- Full queues
  - Routers are forced to have have large queues to maintain high utilizations
  - TCP detects congestion from loss
    - Forces network to have long standing queues in steady-state
- Lock-out problem
  - Drop-tail routers treat bursty traffic poorly
  - Traffic gets synchronized easily → allows a few flows to monopolize the queue space

Active Queue Management

- Design active router queue management to aid congestion control
- Why?
  - Router has unified view of queuing behavior
  - Routers see actual queue occupancy (distinguish queue delay and propagation delay)
  - Routers can decide on transient congestion, based on workload

Design Objectives

- Keep throughput high and delay low
  - High power (throughput/delay)
- Accommodate bursts
- Queue size should reflect ability to accept bursts rather than steady-state queuing
- Improve TCP performance with minimal hardware changes

Lock-out Problem

- Random drop
  - Packet arriving when queue is full causes some random packet to be dropped
- Drop front
  - On full queue, drop packet at head of queue
- Random drop and drop front solve the lock-out problem but not the full-queues problem

Full Queues Problem

- Drop packets before queue becomes full (early drop)
- Intuition: notify senders of incipient congestion
  - Example: early random drop (ERD):
    - If qlen > drop level, drop each new packet with fixed probability *p*
    - Does not control misbehaving users

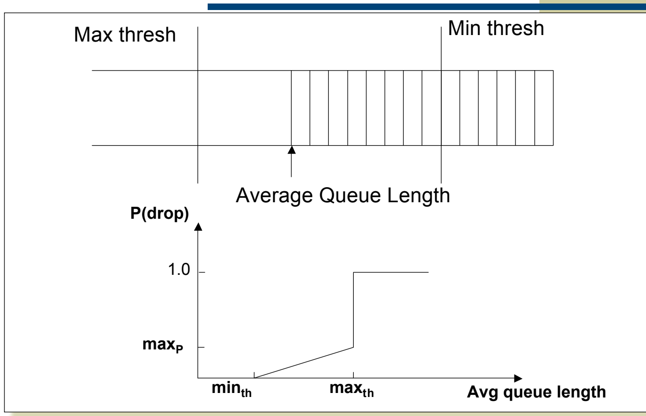
Random Early Detection (RED)

- Detect incipient congestion
- Assume hosts respond to lost packets
- Avoid window synchronization
  - Randomly mark packets
- Avoid bias against bursty traffic

RED Algorithm

- Maintain running average of queue length
- If avg < min_{th} do nothing
  - Low queuing, send packets through
- If avg > max_{th}, drop packet
  - Protection from misbehaving sources
- Else mark packet in a manner proportional to queue length
  - Notify sources of incipient congestion

RED Operation



Overview

- Queue management & RED
- Fair-queuing
- Why QOS?
- Integrated services

Fairness Goals

- Allocate resources fairly
- Isolate ill-behaved users
  - Router does not send explicit feedback to source
  - Still needs e2e congestion control
- Still achieve statistical muxing
  - One flow can fill entire pipe if no contenders
  - Work conserving → scheduler never idles link if it has a packet

What is Fairness?

- At what granularity?
  - Flows, connections, domains?
- What if users have different RTTs/links/etc.
  - Should it share a link fairly or be TCP fair?
- Maximize fairness index?
  - Fairness = $(\Sigma x_i)^2/n(\Sigma x_i^2)$ $0<\text{fairness}<1$
- Basically a tough question to answer – typically design mechanisms instead of policy
  - User = arbitrary granularity

Max-min Fairness

- Allocate user with “small” demand 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

Implementing Max-min Fairness

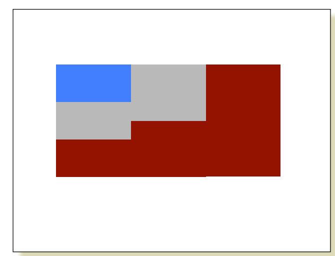
- Generalized processor sharing
  - Fluid fairness
  - Bitwise round robin among all queues
- Why not simple round robin?
  - Variable packet length → can get more service by sending bigger packets
  - Unfair instantaneous service rate
    - What if arrive just before/after packet departs?

Bit-by-bit RR

- Single flow: clock ticks when a bit is transmitted. For packet i:
  - P_i = length, 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
  - Can calculate F_i for each packet if number of flows is know at all times
    - Why do we need to know flow count? → need to know A → This can be complicated

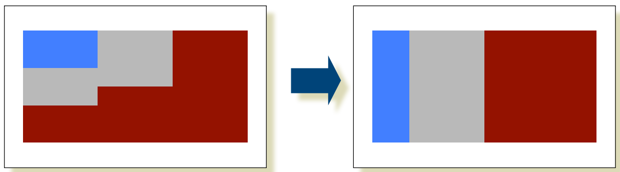
Bit-by-bit RR Illustration

- Not feasible to interleave bits on real networks
  - FQ simulates bit-by-bit RR

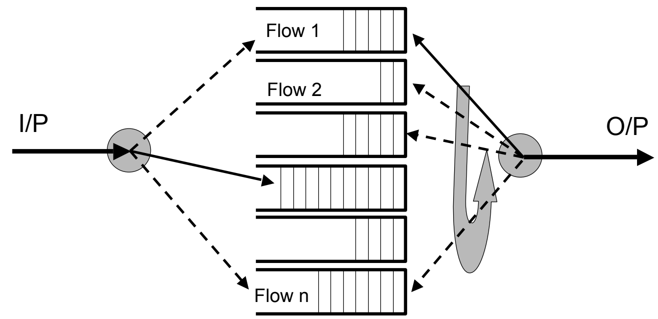


Fair Queuing

- Mapping bit-by-bit schedule onto packet transmission schedule
- Transmit packet with the lowest F_i at any given time
  - How do you compute F_i?

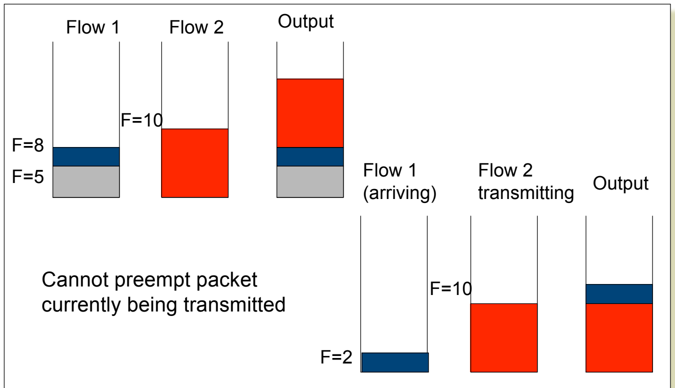


FQ Illustration



Variation: Weighted Fair Queuing (WFQ)

Bit-by-bit RR Example



Fair Queuing Tradeoffs

- FQ can 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
  - dR/dt changes whenever the flow count changes

Overview

- Queue management & RED
- Fair-queuing
- Why QOS?
- Integrated services

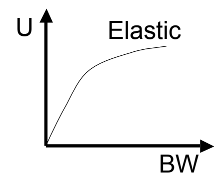
Motivation

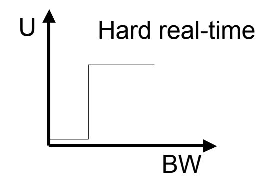
- Internet currently provides one single class of **“best-effort” service**
  - No assurances about delivery
- Existing applications are ***elastic***
  - Tolerate delays and losses
  - Can adapt to congestion
- Future “real-time” applications may be ***inelastic***

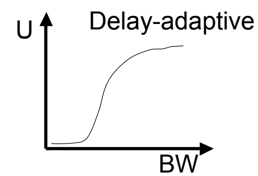
Why a New Service Model?

- What is the **basic objective** of network design?
  - Maximize total bandwidth? Minimize latency?
  - **Maximize user satisfaction** – the total **utility** given to users
- What does utility vs. bandwidth look like?
  - Must be non-decreasing function
  - Shape depends on application

Utility Curve Shapes

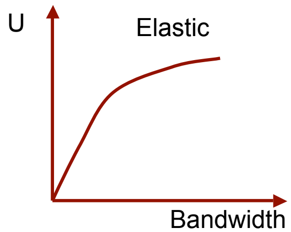






Stay to the right and you are fine for all curves

Utility curve – Elastic traffic

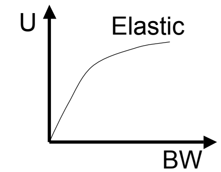


Does equal allocation of bandwidth maximize total utility?

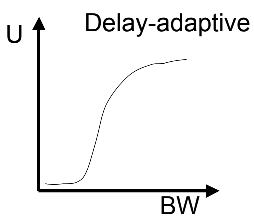
Admission Control

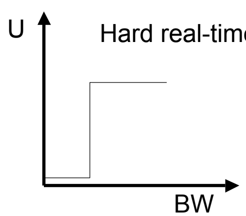
- If U(bandwidth) is concave
  → elastic applications
  - Incremental utility is decreasing with increasing bandwidth
  - Is always advantageous to have more flows with lower bandwidth
    - No need of admission control;

  This is why the Internet works!



Utility Curves – Inelastic traffic





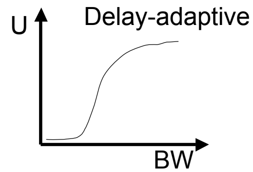
Does equal allocation of bandwidth maximize total utility?

Inelastic Applications

- Continuous media applications
  - **Lower and upper limit** on acceptable performance.
  - BW below which video and audio are not intelligible
  - Internet telephones, teleconferencing with high delay (200 - 300ms) impair human interaction
  - Sometimes called “tolerant real-time” since they can adapt to the performance of the network
- Hard real-time applications
  - Require **hard limits on performance**
  - E.g. control applications

Admission Control

- If U is convex → inelastic applications
  - U(number of flows) is no longer monotonically increasing
  - Need admission control to maximize total utility
- **Admission control** → deciding when adding more people would reduce overall utility
  - Basically avoids overload



Overview

- Queue management & RED
- Fair-queuing
- Why QOS?
- Integrated services

Components of Integrated Services

1. Type of commitment
   What does the network promise?
2. Packet scheduling
   How does the network meet promises?
3. Service interface
   How does the application describe what it wants?
4. Establishing the guarantee
   How is the promise communicated to/from the network
   How is admission of new applications controlled?

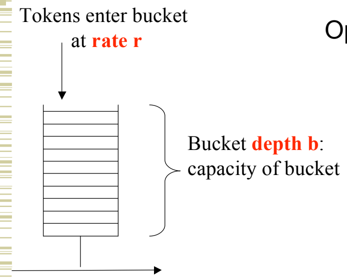
Type of Commitments

- **Guaranteed** service
  - For **hard real-time** applications
  - Fixed guarantee, network meets commitment if clients send at agreed-upon rate
- **Predicted** service
  - For **delay-adaptive** applications
  - Two components
    - If conditions do not change, commit to current service
    - If conditions change, take steps to deliver consistent performance (help apps minimize playback delay)
    - Implicit assumption – network does not change much over time
- **Datagram/best effort service**

Scheduling for Guaranteed Traffic

- Use **token bucket filter** to characterize traffic
  - Described by rate *r* and bucket depth *b*
- Use **Weighted Fair-Queueing** at the routers
- Parekh's bound for worst case queuing delay = b/r

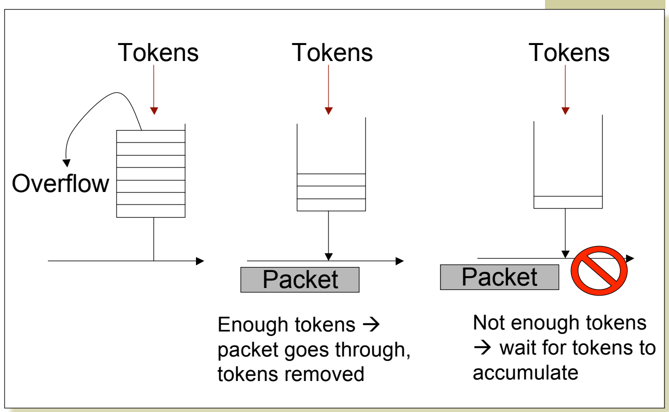
Token Bucket Filter



Operation:

- If bucket fills, tokens are discarded
- Sending a packet of size P uses P tokens
- If bucket has P tokens, packet sent at max rate, else must wait for tokens to accumulate

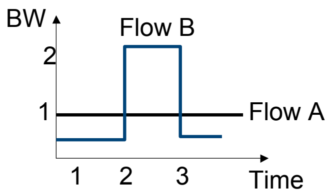
Token Bucket Operation



Token Bucket Characteristics

- On the long run, rate is limited to r
- On the short run, a burst of size b can be sent
- Amount of traffic entering at interval T is bounded by:
  - Traffic = b + r*T
- Information useful to admission algorithm

Token Bucket Specs



Flow A: r = 1 MBps, B=1 byte

Flow B: r = 1 MBps, B=1MB

Guarantee Proven by Parekh

- Given:
  - Flow *i* shaped with token bucket and leaky bucket rate control (depth *b* and rate *r*)
  - Network nodes do WFQ
- Cumulative queuing delay D_i suffered by flow *i* has upper bound
  - **D_i < b/r,** (where r may be much larger than average rate)
  - Assumes that Σr < link speed at any router
  - All sources limiting themselves to *r* will result in no network queuing

Sharing versus Isolation

- Isolation
  - Isolates well-behaved from misbehaving sources
- Sharing
  - Mixing of different sources in a way beneficial to all
- FIFO: sharing
  - each traffic source impacts other connections directly
    - e.g. malicious user can grab extra bandwidth
  - the simplest and most common queueing discipline
  - averages out the delay across all flows
- Priority queues: one-way sharing
  - high-priority traffic sources have impact on lower priority traffic only
  - has to be combined with admission control and traffic enforcement to avoid starvation of low-priority traffic
- WFQ: two-way isolation
  - provides a guaranteed minimum throughput (and maximum delay)

Putting It All Together

- Assume 3 types of traffic: guaranteed, predictive, best-effort
- Scheduling: use WFQ in routers
- Each guaranteed flow gets its own queue
- All predicted service flows and best effort aggregates in single separate queue
  - Predictive traffic classes
    - Worst case delay for classes separated by order of magnitude
    - When high priority needs extra bandwidth – steals it from lower class
  - Best effort traffic acts as lowest priority class

Service Interfaces

- Guaranteed Traffic
  - Host specifies rate to network
  - Why not bucket size b?
    - If delay not good, ask for higher rate
- Predicted Traffic
  - Specifies (r, b) token bucket parameters
  - Specifies delay D and loss rate L
  - Network assigns priority class
  - Policing at edges to drop or tag packets
    - Needed to provide isolation – why is this not done for guaranteed traffic?
      - WFQ provides this for guaranteed traffic

Lessons

- TCP can use help from routers
  - RED → eliminate lock-out and full-queues problems
  - FQ → heavy-weight but explicitly fair to all
- QoS
  - What type of applications are there? → Elastic, hard real-time and adaptive real-time
  - Why do we need admission control → to maximize utility
  - How do token buckets + WFQ provide QoS guarantees?

EXTRA SLIDES

The rest of the slides are FYI

Max-min Fairness Example

- Assume sources 1..n, with resource demands X1..Xn in ascending order
- Assume channel capacity C.
  - Give C/n to X1; if this is more than X1 wants, divide excess (C/n - X1) to other sources: each gets C/n + (C/n - X1)/(n-1)
  - If this is larger than what X2 wants, repeat process

Predicted Service

- FIFO jitter increases with the number of hops
  - Use opportunity for sharing across hops
- FIFO+
  - At each hop: measure average delay for class at that router
  - For each packet: compute difference of average delay and delay of that packet in queue
  - Add/subtract difference in packet header
  - Packet inserted into queues expected arrival time instead of actual
    - More complex queue management!
- Slightly decreases mean delay and significantly decreases jitter

Possible Token Bucket Uses

- Shaping, policing, marking
  - Delay pkts from entering net (shaping)
  - Drop pkts that arrive without tokens (policing)
  - Let all pkts pass through, mark ones without tokens
    - Network drops pkts without tokens in time of congestion

Applications Variations

- Rigid & adaptive applications
  - Rigid – set fixed playback point
  - Adaptive – adapt playback point
    - Gamble that network conditions will be the same as in the past
    - Are prepared to deal with errors in their estimate
    - Will have an earlier playback point than rigid applications
- Tolerant & intolerant applications
  - Tolerance to brief interruptions in service
- 4 combinations

Applications Variations

Really only two classes of applications

1) **Intolerant and rigid**
2) **Tolerant and adaptive**

Other combinations make little sense

3) Intolerant and adaptive
   - Cannot adapt without interruption
4) Tolerant and rigid
   - Missed opportunity to improve delay

So what service classes should the network offer?