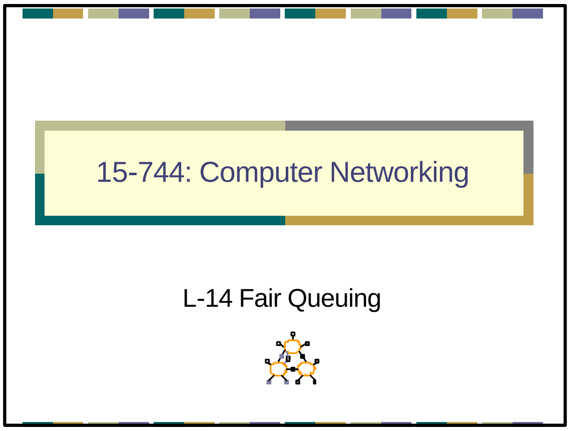
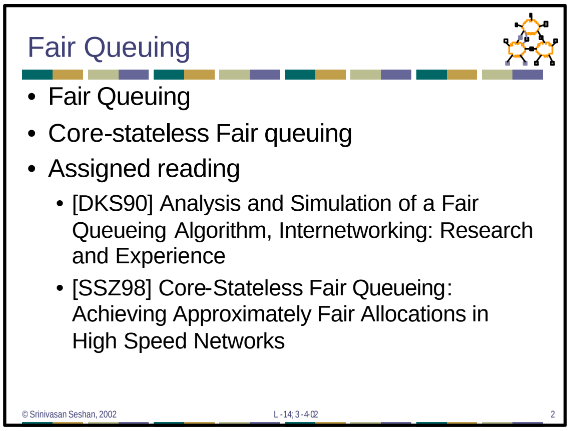


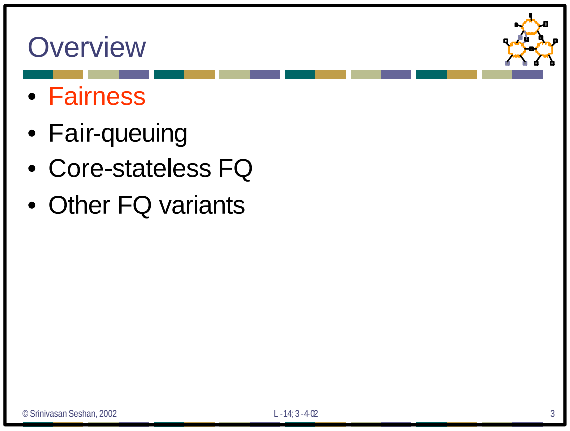
15-744: Computer Networking

L-14 Fair Queuing



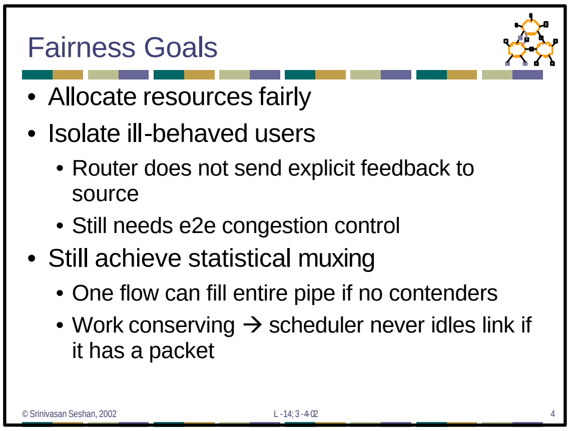
Fair Queuing

- Fair Queuing
- Core-stateless Fair queuing
- Assigned reading
  - [DKS90] Analysis and Simulation of a Fair Queueing Algorithm, Internetworking: Research and Experience
  - [SSZ98] Core-Stateless Fair Queueing: Achieving Approximately Fair Allocations in High Speed Networks



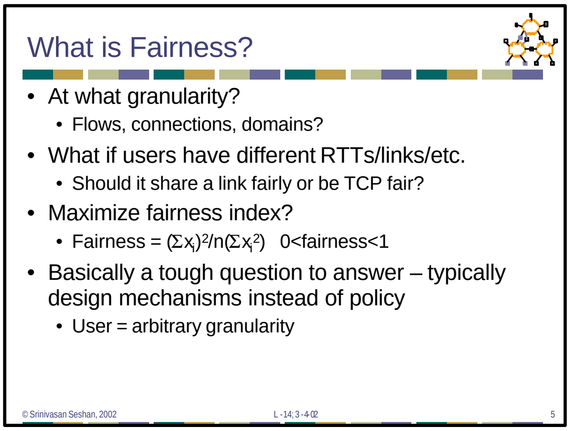
Overview

- Fairness
- Fair-queuing
- Core-stateless FQ
- Other FQ variants



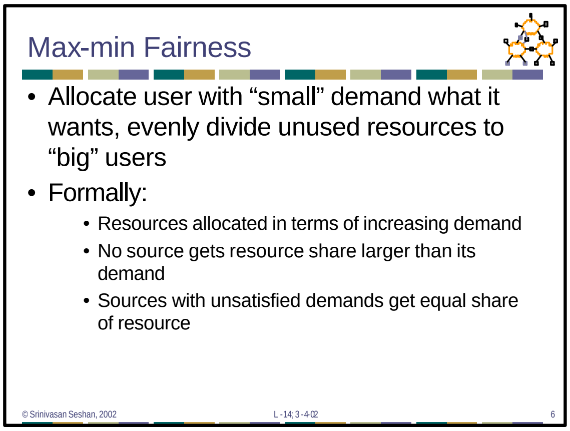
Fairness Goals

- 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

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

Implementing max-min Fairness

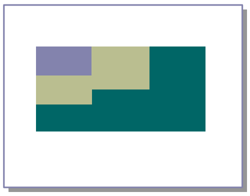
- Generalized processor sharing
  - Fluid fairness
  - Bitwise round robin among all queues
- Why not simple round robin?
  - Variable packet length → can get more service by sending bigger packets
  - Unfair instantaneous service rate
    - What if arrive just before/after packet departs?

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? → This can be complicated

Bit-by-bit RR Illustration

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

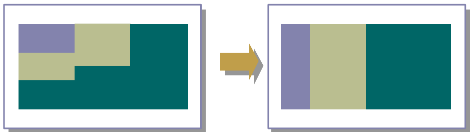


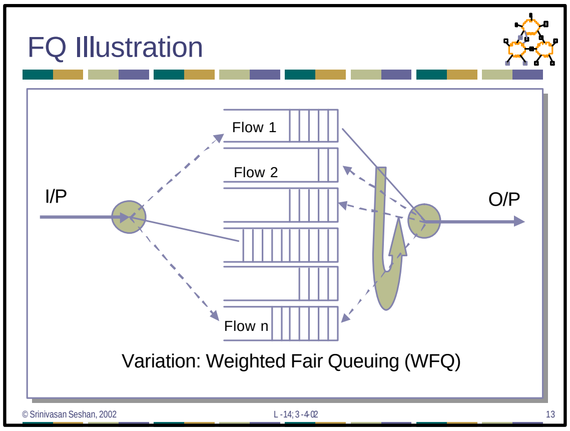
Overview

- Fairness
- Fair-queuing
- Core-stateless FQ
- Other FQ variants

Fair Queuing

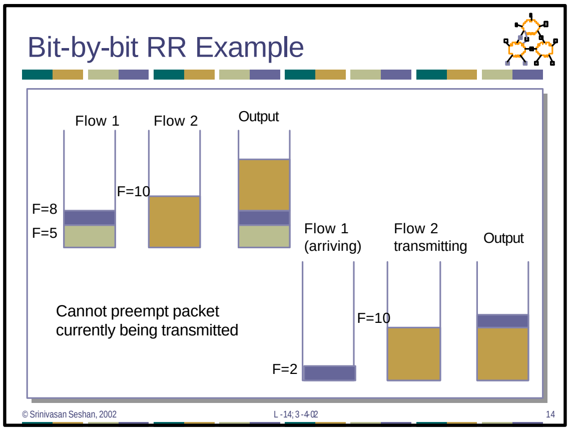
- Mapping bit-by-bit schedule onto packet transmission schedule
- Transmit packet with the lowest F_i at any given time
  - How do you compute F_i?





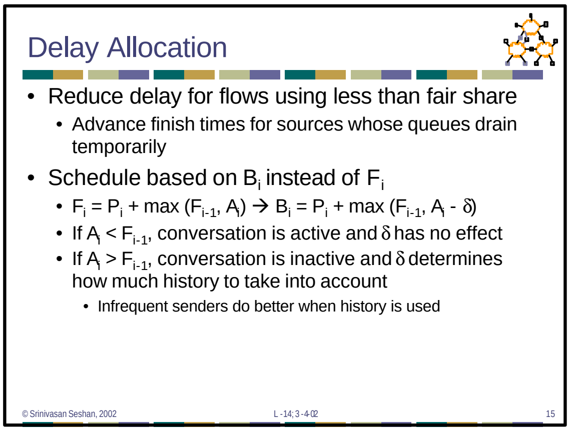
FQ Illustration

Variation: Weighted Fair Queuing (WFQ)



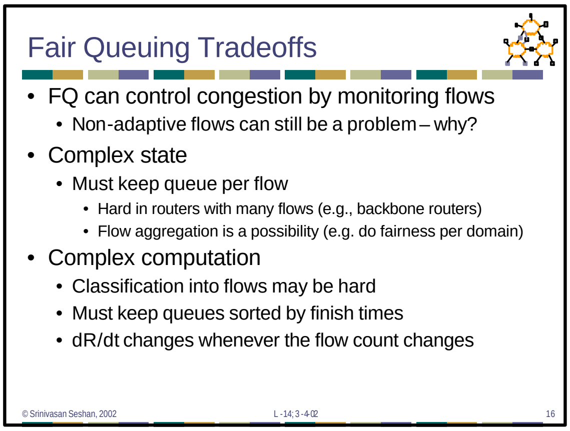
Bit-by-bit RR Example

Cannot preempt packet currently being transmitted



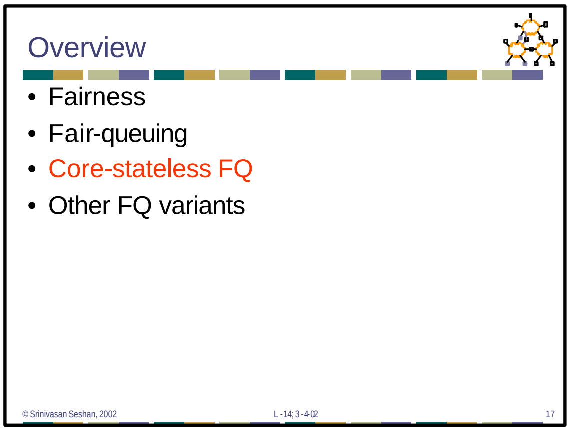
Delay Allocation

- Reduce delay for flows using less than fair share
  - Advance finish 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 $A_i < F_{i-1}$, conversation is active and δ has no effect
  - If $A_i > F_{i-1}$, conversation is inactive and δ determines how much history to take into account
    - Infrequent senders do better when history is used



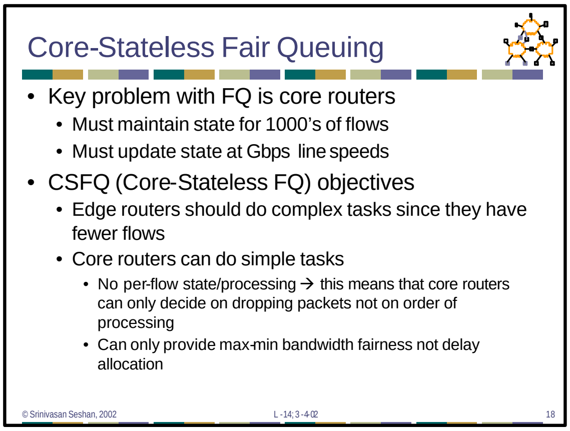
Fair Queuing Tradeoffs

- FQ can 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

- Fairness
- Fair-queuing
- Core-stateless FQ
- Other FQ variants



Core-Stateless Fair Queuing

- Key problem with FQ is core routers
  - Must maintain state for 1000's of flows
  - Must update state at Gbps line speeds
- CSFQ (Core-Stateless FQ) objectives
  - Edge routers should do complex tasks since they have fewer flows
  - Core routers can do simple tasks
    - No per-flow state/processing → this means that core routers can only decide on dropping packets not on order of processing
    - Can only provide max-min bandwidth fairness not delay allocation

Core-Stateless Fair Queuing

- Edge routers keep state about flows and do computation when packet arrives
- DPS (Dynamic Packet State)
  - Edge routers label packets with the result of state lookup and computation
- Core routers use DPS and local measurements to control processing of packets

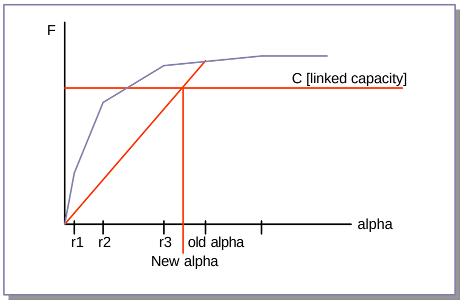
Edge Router Behavior

- Monitor each flow i to measure its arrival rate (r_i)
  - EWMA of rate
  - Non-constant EWMA constant
    - $e^{-T/K}$ where T = current interarrival, K = constant
    - Helps adapt to different packet sizes and arrival patterns
- Rate is attached to each packet

Core Router Behavior

- Keep track of fair share rate α
  - Increasing α does not increase load (F) by N * α
  - $F(\alpha) = \Sigma_i \min(r_i, \alpha)$ → what does this look like?
  - Periodically update α
  - Keep track of current arrival rate
    - Only update α if entire period was congested or uncongested
- Drop probability for packet = $\max(1- \alpha/r, 0)$

F vs. Alpha



Estimating Fair Share

- Need $F(\alpha)$ = capacity = C
  - Can't keep map of $F(\alpha)$ values → would require per flow state
  - Since $F(\alpha)$ is concave, piecewise-linear
    - $F(0) = 0$ and $F(\alpha)$ = current accepted rate = F_c
    - $F(\alpha) = F_c / \alpha$
    - $F(\alpha_{new}) = C$ → $\alpha_{new} = \alpha_{old} * C/F_c$
- What if a mistake was made?
  - Forced into dropping packets due to buffer capacity
  - When queue overflows α is decreased slightly

Other Issues

- Punishing fire-hoses – why?
  - Easy to keep track of in a FQ scheme
- What are the real edges in such a scheme?
  - Must trust edges to mark traffic accurately
  - Could do some statistical sampling to see if edge was marking accurately

Overview

- Fairness
- Fair-queuing
- Core-stateless FQ
- Other FQ variants

Stochastic Fair Queuing

- Similar idea as Stochastic Fair Blue
  - Compute a hash on each packet
  - Instead of per-flow queue have a queue per hash bin
  - An aggressive flow steals traffic from other flows in the same hash
- Queues serviced in round-robin fashion
  - Has problems with packet size unfairness
- Memory allocation across all queues
  - When no free buffers, drop packet from longest queue

Deficit Round Robin

- Each queue is allowed to send Q bytes per round
- If Q bytes are not sent (because packet is too large) deficit counter of queue keeps track of unused portion
- If queue is empty, deficit counter is reset to 0
- Uses hash bins like Stochastic FQ
- Similar behavior as FQ but computationally simpler

Self-clocked Fair Queuing

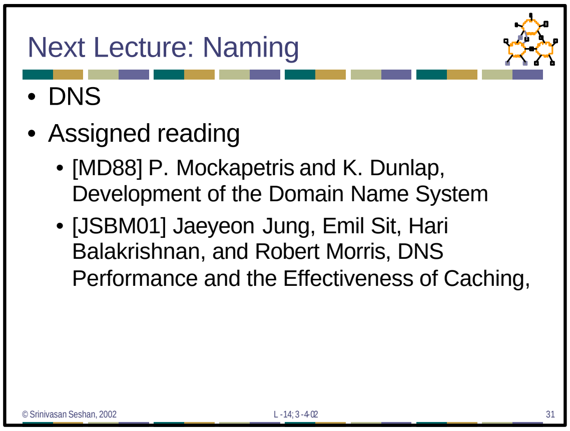
- Virtual time to make computation of finish time easier
- Problem with basic FQ
  - Need be able to know which flows are really backlogged
    - They may not have packet queued because they were serviced earlier in mapping of bit-by-bit to packet
    - This is necessary to know how bits sent map onto rounds
    - Mapping of real time to round is piecewise linear → however slope can change often

Self-clocked FQ

- Use the finish time of the packet being serviced as the virtual time
  - The difference in this virtual time and the real round number can be unbounded
- Amount of service to backlogged flows is bounded by factor of 2

Start-time Fair Queuing

- Packets are scheduled in order of their start not finish times
- Self-clocked → virtual time = start time of packet in service
- Main advantage → can handle variable rate service better than other schemes
  - Useful for hierarchical schedulers



Next Lecture: Naming

- DNS
- Assigned reading
  - [MD88] P. Mockapetris and K. Dunlap, Development of the Domain Name System
  - [JSBM01] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris, DNS Performance and the Effectiveness of Caching,