CS 514: Computer Networks
Lecture 7: Fair Queuing

Xiaowei Yang
xwy@cs.duke.edu
• A fundamental question of networking: who gets to send at what speed?
Router-Assisted Congestion Control

- Active Queue Management
- Signaling
- Scheduling
Fair Queuing
Fair Queuing Motivation

• End-to-end congestion control + FIFO queue (or AQM) has limitations
  – What if sources mis-behave?

• Fair Queuing
  – A queuing algorithm that aims to “fairly” allocate buffer, bandwidth, latency among competing users
Outline

• What is fair?
• Weighted Fair Queuing
• Other FQ variants
What is fair?

• Fair to whom?
  – Source, Receiver, Process
  – Flow / conversation: Src and Dst pair
    • Flow is considered the best tradeoff

• Maximize fairness index?
  – Fairness = \( \frac{(\sum x_i)^2}{n(\sum x_i^2)} \) \( 0 < \text{fairness} < 1 \)

• What if a flow uses a long path?

• Tricky, no satisfactory solution, policy vs mechanism
One definition: Max-min fairness

- Many fair queuing algorithms aim to achieve this definition of fairness
- Informally
  - Allocate user with “small” demand what it wants, evenly divide unused resources to “big” users
- Formally
  - 1. No user receives more than its request
  - 2. No other allocation satisfies 1 and has a higher minimum allocation
    - Users that have higher requests and share the same bottleneck link have equal shares
  - Remove the minimal user and reduce the total resource accordingly, 2 recursively holds
Max-min example

1. Increase all flows’ rates equally, until some users’ requests are satisfied or some links are saturated.
2. Remove those users, reduce the resources, and repeat step 1.

- Assume sources 1...n, with resource demands X1...Xn in an 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.
An example

• A 10Mbps bottleneck link

• 3 Users: r1: 1Mbps, r2: 6Mbps, r3: 8Mbps
  – 1 Mbps, 4.5Mbps, 4.5Mbps

• 4 Users: r1, 1Mbps, r2: 2Mbps, r3: 6Mbps, r4: 8Mbps
  – 1 Mbps, 2 Mbps, 3.5 Mbps, 3.5Mbps
Design of weighted fair queuing

• Resources managed by a queuing algorithm
  – Bandwidth: Which packets get transmitted
  – Promptness: When do packets get transmitted
  – Buffer: Which packets are discarded
  – Examples: FIFO
    • The order of arrival determines all three quantities
Design goals

• Max-min fair
• Work conserving: link’s not idle if there is work to do
• Isolate misbehaving sources
• Has some control over promptness
  – E.g., lower delay for sources using less than their full share of bandwidth
A simple fair queuing algorithm

- Nagle’s proposal: separate queues for packets from each individual source
- Different queues are serviced in a round-robin manner
- Limitations
  - Is it fair?
  - What if a packet arrives right after one departs?
Implementing max-min Fairness

- Generalized processor sharing
  - Fluid fairness
  - Bitwise round robin among all queues

- WFQ:
  - Emulate this reference system in a packetized system
  - Challenges: bits are bundled into packets. Simple round robin scheduling does not emulate bit-by-bit round robin
Emulating Bit-by-Bit round robin

- Define a virtual clock: the round number $R(t)$ as the number of rounds made in a bit-by-bit round-robin service discipline up to time $t$

- A packet with size $P$ whose first bit serviced at round $R(t_0)$ will finish at round: $R(t) = R(t_0) + P$

- Schedule which packet gets serviced based on the finish round number
Example

- First packets from all three flows arrive at the same time
- Second green arrives at virtual clock 4
Compute finish times

• Arrival time of packet $i$ from flow $\alpha$: $t_{i}^{\alpha}$
• Packet size: $P_{i}^{\alpha}$
• $S_{i}^{\alpha}$ be the round number when the packet starts service
• $F_{i}^{\alpha}$ be the finish round number
• $F_{i}^{\alpha} = S_{i}^{\alpha} + P_{i}^{\alpha}$
• $S_{i}^{\alpha} = \text{Max} \ (F_{i-1}^{\alpha}, R(t_{i}^{\alpha}))$
Virtual Clock Ticks at Different Rates

- Single flow: clock ticks when a bit is transmitted. For packet i:
  - Round number $\leq$ Arrival time $A_i$
  - $F_i = S_i + P_i = \max(F_{i-1}, A_i) + P_i$

- Multiple flows: clock ticks when a bit from all active flows is transmitted
  - When the number of active flows vary, clock ticks at different speed: $\frac{\partial R}{\partial t} \propto \frac{1}{N_{ac}(t)}$

- Finish times do not change once computed
unit link speed
Flow 1: pkt size 2
Flow 2: pkt size 1
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 $\delta$ has no effect
  – If $A_i > F_{i-1}$, conversation is inactive and $\delta$ determines how much history to take into account
    • Infrequent senders do better when history is used
  – When $\delta = 0$, no effect
  – When $\delta = \infty$, an infrequent sender preempts other senders
Weighted Fair Queuing

- Different queues get different weights
  - Take $w_i$ amount of bits from a queue in each round
  - $F_i = S_i + P_i / w_i$

\[ \begin{align*}
  &w=1 \\
  &\text{Green} \\
  &w=2 \\
  &\text{Blue}
\end{align*} \]
Outline

• What is fair?
• Weighted Fair Queuing
• Other FQ variants
Stochastic Fair Queuing

• Goal: fixed number of queues rather than various number of queues
  – Compute a hash on each packet
  – Instead of per-flow queue have a queue per hash bin
  – Queues serviced in round-robin fashion
  – Memory allocation across all queues
  – When no free buffers, drop packet from longest queue

• Limitations
  – An aggressive flow steals traffic from other flows in the same hash
  – Has problems with packet size unfairness
Deficit Round Robin

- O(1) rather than O(log Q)
- 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
• Unused quantum is saved for the next round to offset packet size unfairness
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
CSFQ architecture

- Island of routers
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: $r_i^{\text{new}} = (1 - e^{-T_i^k/K}) \frac{l_i^k}{T_i^k} + e^{-T_i^k/K} r_i^{\text{old}}$,
  - New technique: 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 $\alpha(t)$
  – Increasing $\alpha(t)$ does not increase load ($F$) by $N \times \alpha(t)$ (max-min fairness)
  – $F(\alpha(t)) = \sum_{i=1,n} \min(r_i, \alpha(t)) \rightarrow$ what does this look like?
  – Periodically update $\alpha(t)$
  – Keep track of current arrival rate
    • Only update $\alpha(t)$ if an entire period was congested or uncongested

• Drop probability for packet
  – $\max(1 - \alpha(t)/r_i, 0)$
  – If $r >$ target rate $\alpha(t)$, then $r$ becomes $\alpha(t)$ after drops
F vs. Alpha

\[
\begin{align*}
F & \text{ vs. } \alpha \\
& \text{C [link capacity]} \\
& \text{r1, r2, r3} \\
& \text{old alpha, new alpha}
\end{align*}
\]
Summary

• Fair queuing
  – Max-min fair
  – Weighted fair queuing
  – Stochastic fair queuing
  – Deficit round robin
  – Core stateless fair queuing
  – Not covered: start time fair queuing