BITS Pilani
Pilani Campus

Advance Computer Networks (CS G525)

Virendra S Shekhawat
Department of Computer Science and Information Systems

innovate achieve lead

BITS Pilani
Pilani Campus

First Semester 2018-2019
Slide_Deck_M3_3

Agenda

- Congestion Control at Routers-Queuing Algorithms
  - Fair Queuing (FQ)
  - Nagle's FQ Algorithm
  - Max-Min Fairness
  - Weighted Fair Queuing (WFQ)
  - Other Queuing Algorithms (FIFO, CSFQ, RED)
- Reading
  - Random Early Detection Gateways for Congestion Avoidance by Sally Floyd 1993

Congestion Control: Queuing Algorithms

- Congestion can be controlled at gateways through routing and queuing algorithms
- What does queuing algorithms…?
  - Controls the order in which packets are sent
  - Usages of the gateways buffer space
  - Important:
    - Do not change the total traffic on gateways outgoing links
    - Affects the collective behavior of flow control algorithms

Queuing Algorithms Functions

- Which packet get transmitted? **(Bandwidth)**
- When do these packets get transmitted? **(Promptness)**
- Which and when packets get discarded by the gateway? **(Buffer Space)**
- Queuing also affects the latency….!

Packet Drop Dimensions

Aggregation

Per-connection state — Class-based queuing — Single class

Drop position

Head — Random location — Tail

Early drop — Overflow drop

Fairness among Flows

- At what granularity?
  - s-d pair, source base, receiver base, process base?
- What if users have different RTTs/links/etc.
  - Should it share a link fairly or be TCP fair?
- Maximize fairness index?
  - Fairness = $(\Sigma x_i)^2/n(\Sigma x_i^2)$ $0<\text{fairness}<1$

Fairness Goals For Routers

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

Nagle's FQ Algorithm

- Gateway maintains separate queues for packets from each individual source
  - Queues are serviced in Round Robin manner
  - Prevents source from sending packet too quickly
    - It increases its own queue length only!

- Drawback
  - Packet size was not considered
    - Source using large packets get more bandwidth

Max-min Fairness

- Allocate user with “small” demand what it wants, evenly divide unused resources to “big” users
- Formally:
    a) Resources allocated in terms of increasing demand
    b) No source gets resource share larger than its demand
    c) Sources with unsatisfied demands get equal share of resource

Max-min Fairness Algorithm

- Assume sources 1....n, with resource demands X_1....X_n in ascending order
- Assume channel capacity is C
  - Give C/n to X_1; if this is more than X_1 wants, divide excess (C/n - X_1) to other sources: each gets C/n + (C/n - X_1)/(n-1)
  - If this is larger than what X_2 wants, repeat the process

How to Implement max-min Fairness…?

- Generalized processor sharing
  - Fluid fairness
  - Bitwise round robin among all queues
  - Practical feasibility…?

- Why not simple round robin?
  - Variable packet length → can get more service by sending bigger packets
  - Unfair instantaneous service rate
    - Packets arrive just before/after packet departs?

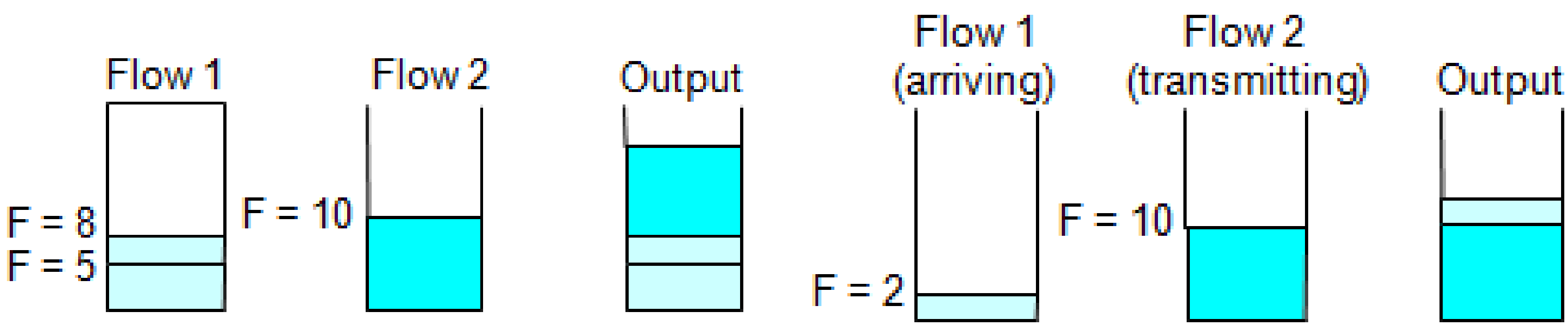
FQ Algorithm

innovate achieve lead

- Suppose clock ticks each time a bit is transmitted
- Let P_i denote the length of packet *i*
- Let S_i denote the time when start to transmit packet *i*
- Let F_i denote the time when finish transmitting packet *i*
- $F_i = S_i + P_i$
- When does router start transmitting packet *i*?
  - If packet i arrives before router finished packet *i* - 1 from this flow, then immediately after last bit of *i* - 1 (F_{i-1})
  - If no current packets for this flow, then start transmitting when arrives (call this A_i)
- Thus: $F_i = MAX\ (F_{i-1}, A_i) + P_i$

FQ Algorithm (cont...)

- For multiple flows
  - Calculate F_i for each packet that arrives on each flow
  - Treat all F_i's as time stamps
  - Next packet to transmit is one with lowest timestamp
- Not perfect: can't preempt current packet
- Example



Fair Queuing Tradeoffs

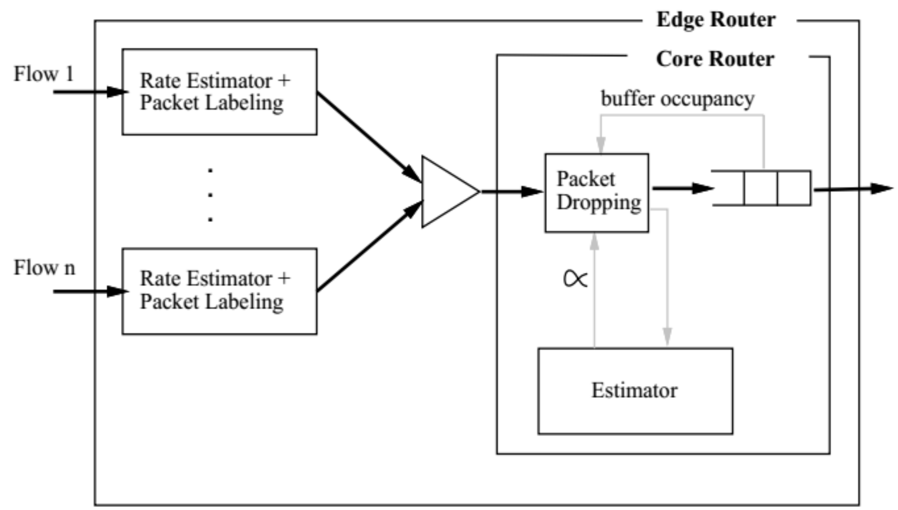
- FQ can control congestion by monitoring flows
  - Non-adaptive flows can still be a problem – why?
- Complex state
  - Must keep queue per flow
    - Hard in routers with many flows (e.g., backbone routers)
    - Flow aggregation is a possibility (e.g. do fairness per domain)
- Complex computation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

Core-Stateless Fair Queuing

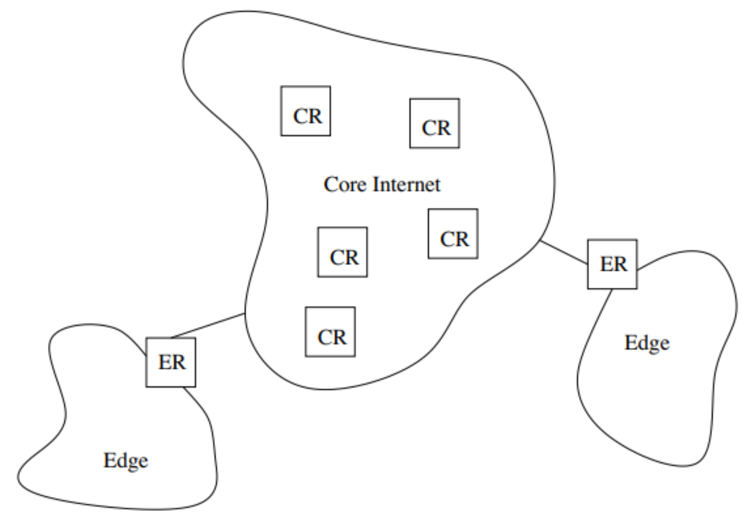
- Key problem with FQ is core routers
  - Must maintain state for 1000's of flows
  - Must update state at Gbps line speeds
- CSFQ (Core-Stateless FQ) objectives
  - Edge routers should do complex tasks since they have fewer flows
    - Maintains per flow state
  - Core routers can do simple tasks
    - Core routers can only decide on dropping packets based on the level of congestion

Core-Stateless Fair Queuing

- Edge Router
  - Maintains Flow per state.
  - Label the packet with the rate estimate



- Core Router
  - Uses FIFO Queuing
  - Probabilistic dropping depending on label, fair rate of router depending on aggregate traffic



Edge Router Behavior

- Monitor each flow i to measure its arrival rate (r_i)
  - EWMA of rate $$r_i^{new} = (1 - e^{-T_i^k/K})\frac{l_i^k}{T_i^k} + e^{-T_i^k/K} r_i^{old}$$
    Here $T_i^k = t_i^k - t_i^{k-1}$
  - Non-constant EWMA constant
    - $e^{-T/K}$ where T = current inter-arrival, K = constant
    - Helps to adapt different packet sizes and arrival patterns
- Rate is attached to each packet

Core Router Behavior

- Keep track of fair share rate 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 = $\max(1- \alpha/r, 0)$

FIFO + DropTail

- Widely used 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 the edges (e.g., TCP)
- Does not separate between different flows
- No policing: send more packets → get more service
- Synchronization Problem
  – When queue overflow, packet dropped by gateway for almost all flows hence hosts react to same events

Internet Queuing Problems

- Full queues
  - Routers are forced to have large queues to maintain high utilizations
  - TCP detects congestion from loss
    - Forces network to have long standing queues in steady-state which increases average delay in the network

- Lock-out problem
  - Drop-tail routers treat bursty traffic poorly
  - Traffic gets synchronized easily → allows a few flows to monopolize the queue space (prevent new flows to enter the queue)

Lock-out Problem: Solution

- 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: Solution

- Drop packets before queue becomes full (early drop)
- ***Intuition:*** notify senders of incipient congestion

- Example: Early Random Drop (ERD):
  - If queue length > drop level, drop each new packet with fixed probability *p*
  - Will this control misbehaving users…?

Active Queue Management (AQM)

- Design active router queue management to aid congestion control
- Why?
  - Routers can distinguish between propagation and persistent queuing delays
  - Routers can decide on transient congestion, based on workload

Active Queue Designs

- Solutions that requires to modify both router and hosts
  - DECbit: congestion bit in packet header
  - Calculates the **avg queue length** for last cycle (busy+idle) plus the current busy period
  - When avg queue length exceeds → set the congestion bit in the header of the arriving packets
  - If less than 50% of last window's worth had bit set
    - Increase `CongestionWindow` by 1 packet
  - If 50% or more of last window's worth had bit set
    - Decrease `CongestionWindow` by 0.875 times
- Solutions that requires to modify only router and host uses TCP
  - Fair queuing
    - Per-connection buffer allocation
  - RED (Random Early Detection)
    - Drop packet or set bit in packet header as soon as congestion is starting

Design Objectives: Queuing Algorithms

- Keep throughput high and delay low
- Accommodate bursts to avoid bias against bursty traffic
- Queue size should reflect ability to accept bursts rather than steady-state queuing
- Maintain an upper bound on the average queue length even in the absence of cooperation from TP protocols
- Avoidance of global synchronization

RED Algorithm

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

RED Algorithms

- Two separate algorithms
  - One for computing the average queue size
    - To determine the degree of burstiness
  - Second for calculating packet marking probability
    - Gives current level of congestion
      - More packet are marked indicate higher congestion

Queue Estimation

- Standard EWMA: avgq = $(1\text{-}w_q)$ avgq + $w_q$qlen
  - Special fix for idle periods (empty queue)– why?
  - Assume m packets of small size processed during idle period
- Upper bound on w_q depends on min_{th}
  - Want to ignore transient congestion
  - Set w_q such that certain burst size does not exceed min_{th}
- Lower bound on w_q is to detect congestion relatively quickly
- Typical $w_q = 0.002$

Thresholds

- min_{th} determined by the utilization requirement
  - Tradeoff between queuing delay and utilization
- Relationship between max_{th} and min_{th}
  - Want to ensure that feedback has enough time to make difference in load
  - Depends on average queue increase in one RTT
  - Paper suggests ratio of two

Packet Marking

- Marking probability is based on queue length
  - $P_b = max_p(avgq - min_{th}) / (max_{th} - min_{th})$
- Just marking based on P_b can lead to clustered marking
  - Could result in synchronization
  - Better to bias P_b by history of unmarked packets
  - $P_a = P_b/(1 - count*P_b)$
  - This ensures gateway doesn't wait too long to mark a packet

RED Algorithm

```
Initialization:
    avg ← 0
    count ← −1
for each packet arrival
    calculate new avg. queue size avg:
        if the queue is nonempty
            avg ← (1 − w_q)avg + w_q q
        else
            m ← f(time − q_time)
            avg ← (1 − w_q)^m avg
    if min_th ≤ avg < max_th
        increment count
        calculate probability p_a:
            p_b ←
                max_p(avg − min_th)/(max_th − min_th)
            p_a ← p_b/(1 − count · p_b)
        with probability p_a:
            mark the arriving packet
            count ← 0
    else if max_th ≤ avg
        mark the arriving packet
        count ← 0
    else count ← −1
when queue becomes empty
    q_time ← time
```

Where: $avg \leftarrow (1 - w_q)avg + w_q\, q$, $avg \leftarrow (1 - w_q)^m avg$, $p_b \leftarrow max_p(avg - min_{th})/(max_{th} - min_{th})$, $p_a \leftarrow p_b/(1 - count \cdot p_b)$.

RED - Summary

- Detect incipient congestion, allow bursts
- Keeps power (throughput/delay) high
  - Keep average queue size low
  - Assume hosts respond to lost packets
- Avoids window synchronization
  - Randomly mark packets
- Avoids bias against bursty traffic
- Some protection against ill-behaved users

Extending RED for Flow Isolation

- Problem: what to do with non-cooperative flows?
- Fair queuing achieves isolation using per-flow state – expensive at backbone routers
  – How can we isolate unresponsive flows without per-flow state?
- RED penalty box
  – Monitor history for packet drops, identify flows that use disproportionate bandwidth
  – Isolate and punish those flows

Thank You!