Lecture 20 Overview

- Router buffer Management
  - FIFO
  - RED

- Router traffic Policing/Scheduling
Typical high-performance router

- IQ + VoQ + OQ
  - Speedup of 2
  - Central scheduler
  - Fixed-sized internal cells

- Pro
  - Can achieve utilization of 1
  - Can scale to > Tb/s

- Con
  - Multiple congestion points
  - Complexity
Key Router Challenges

- **Buffer management**: which packet to drop when?
  - We only have finite-length queues
- **Scheduling**: which packet to transmit next?
Basic Buffer Management

- 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
Leaves responsibility of preventing congestion completely to the edges
- Transport protocols will need to make sure queues in routers are not so full that they drop, (called congestion control)

- Does not separate between different flows
  - Packet can be dropped regardless of what TCP or UDP flow it is part of

- **No policing**: send more packets \(\rightarrow\) get more service

- Synchronization: end hosts react to same problems
Active Queue Management

- Design active router queue management to aid in reducing the congestion at routers

- 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

- Help with transport protocol (e.g., TCP) performance with minimal hardware changes in router
Random Early Detection

- Detect incipient congestion

- **Assume hosts respond to lost packets**
  - We know they will retransmit based on losses
  - Soon we will see losses will also make them slow down!

- Avoid window synchronization
  - Randomly mark packets

- Avoid bias against bursty traffic
RED Algorithm

- Maintain running average of queue length in router

- If $\text{avg} < \text{min}_{th}$ do nothing
  - Low queuing, send packets through

- If $\text{avg} > \text{max}_{th}$, drop packet
  - Protection from misbehaving sources

- Else drop/mark packet in a manner proportional to queue length
  - Notify sources of incipient congestion
  - Dropping vs Marking tradeoff (Explicit Congestion Notification)
RED Operation

Max thresh  Min thresh

Average Queue Length

P(drop)

1.0

maxp

minth  maxth

Avg queue length
So far we’ve done flow-based **traffic policing**
- Limit the rate of one flow regardless of the load in the network

In general, need **scheduling**
- Dynamically allocate resources when multiple flows compete
- Give each “flow” (or src/destination pair) own queue (at least theoretically)

**Weighted fair queuing**
- Proportional-share scheduling
- Schedule round-robbins among queues in proportion to some weight parameter
Example with contending hosts

1 UDP (10 Mbps) and 31 TCPs sharing a 10 Mbps line
UDP vs. TCP w/FIFO
TCP vs. UDP w/Fair Queuing

![Graph showing throughput (Mbps) vs. flow number with bars for each flow. The bar graph demonstrates the throughput for different flow numbers across the range of 1 to 32, with throughput values ranging from 0.20 to 0.40 Mbps.]
(Weighted) Fair Queuing
Fair Queuing

- Maintain a queue for each flow
  - What is a flow?
    » We approximate this with the 5 tuple
    » (IP src, IP dst, port src, port dst, protocol)

- Implements max-min fairness: each flow receives
  \[ \text{min}(r_i, f) \] , where
  - \( r_i \) – flow arrival rate
  - \( f \) – link fair rate (see next slide)

- Weighted Fair Queuing (WFQ) – associate a weight with each flow to divvy bandwidth up non-equally
Fair Rate Computation

- If link congested, compute $f$ such that

$$\sum_i \min(r_i, f) = C$$

$f = 4$:
- $\min(8, 4) = 4$
- $\min(6, 4) = 4$
- $\min(2, 4) = 2$
Another Example

- Associate a weight $w_i$ with each flow $i$
- If link congested, compute $f$ such that

$$\sum_i \min(r_i, f \times w_i) = C$$

$$(w_1 = 3) \quad 8$$
$$(w_2 = 1) \quad 6$$
$$(w_3 = 1) \quad 2$$

$f = 2$:
- $\min(8, 2*3) = 6$
- $\min(6, 2*1) = 2$
- $\min(2, 2*1) = 2$

Flow $i$ is guaranteed to be allocated a rate $\geq w_i C / (\Sigma_k w_k)$

If $\Sigma_k w_k \leq C$, flow $i$ is guaranteed to be allocated a rate $\geq w_i$
Fluid Flow

- Simplification: Flows can be served one bit at a time

- WFQ can be implemented using bit-by-bit weighted round robin
  - During each round from each flow that has data to send, send a number of bits equal to the flow’s weight
Fluid Flow Example

- Red flow has packets backlogged between time 0 and 10
- Other flows have packets continuously backlogged
- All packets have the same size
 Packet-Based Implementation

- Packet (Real) system: packet transmission cannot be preempted. Why?

- Issue: flows can get more bandwidth by sending bigger packets

- Solution: serve packets in the order in which they would have finished being transmitted in the fluid flow system
Packet-Based Example

- Select the first packet that finishes in the fluid flow system

Service in fluid flow system

Packet system

time
For next time…

- Stopping congestion at hosts instead of routers
  - Read Ch. 6.3 in P&D