Packet Scheduling

- Need for Packet Scheduling
  - Best Effort Service: Fairness
  - Guaranteed Service: Flow isolation and resource (bandwidth) reservations

- Requirements
  - Implementation Simplicity
    - Run at line speed: 320 nsec for minimum IP pkt on Gbps link
    - Want O(1) time for scheduling
    - Limitation: Amount of memory required for scheduling state
  - Best Effort Fairness and Guaranteed Service Isolation
    - Allocation is fair if it satisfies the max-min allocation criterion
    - Isolation: Flow i’s misbehavior does not affect other flows
  - Derivable Performance Bounds
  - Efficient Admission Control

FIFO (FCFS) Queueing Discipline

- Packets are transmitted in arrival order
- Tail drop
  - Drop any pkt arriving to a full queue (buffer)
- FIFO + Tail drop
  - Simplest scheduling policy + drop policy
  - Most widely used in Internet routers
  - Pushes congestion control responsibility to end systems

FIFO Service Trajectory

- A backlog develops because of temporary overload
  - \[ B(t) = A(t) - D(t) \]
The Need for Fair Queueing

- FIFO queueing depends on TCP-like congestion control at the endsystems
  - "Unfriendly" flows can take most of the bandwidth from behaving flows
- Fair Queueing (FQ)
  - Maintain a queue for each flow at a router
  - Serve the queues in round-robin manner
  - Flow Isolation (Segregation)
    - A greedy flow will fill up its allocation of buffers and excessive pkts from G will get dropped
    - Well-behaved flows are protected from flow G's greediness
- Complications
  - Unequal length packets
  - Supporting flow priorities

GPS Example 1

If instead, a 4 unit pkt from Flow 1 arrives at t=0:
- It would finish 1 unit of the pkt at t=4 and the rest by t=8

GPS Example 2

Suppose:
- Equal weights: w(i) = 1, all i
- t=0: Pkts of size 1, 2, and 2 arrive from flows A, B, and C
- t=4: Pkt of size 2 arrives from flow A
- R = 1 unit/sec

Generalized Processor Sharing

- Ideal Packet Scheduler
  - Work-conserving (Send pkts as long as there is a backlog)
  - Achieves a max-min fair allocation
- Service Model
  - Each packet has its own separate queue
  - Visit each queue in round-robin fashion
  - Infinitesimally small amount of service in each visit
- The GPS model actually assumes a fluid model
  - Infinitely small transmission units
  - Not realizable
    - A realizable scheduler transmits whole packets
**Weighted Fair Queueing (1)**

<table>
<thead>
<tr>
<th>Real Scheduler</th>
<th>Virtual Scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Packet With</td>
<td>Compute Finish Number</td>
</tr>
<tr>
<td>Minimum Finish Number</td>
<td>When Packet Arrives</td>
</tr>
<tr>
<td>Earliest Finish Number First (Packet-By-Packet)</td>
<td>Round Robin (e.g., Byte-By-Byte)</td>
</tr>
</tbody>
</table>

- The real scheduler only cares about the relative order of finish numbers (virtual finish time).
  - Doesn’t care about round numbers (virtual time).
- The virtual scheduler emulates GPS.
  - Computes finish numbers for each packet.
  - Packet completes service when the round number increases beyond the finish time.

**WFQ (2)**

- The Finish Number of a Packet
  - Is a virtual time that is used only to order packet service (has no other meaning).
  - Is different than its finish time in the real world.
  - Does not change after it is computed for a packet (e.g., future pkt arrivals/departures).
- Round Number (Virtual Time)
  - Needed by the virtual scheduler to compute finish number of an arriving packet.
  - Increases at a rate inversely proportional to the sum of weights of active flows in GPS.
    - \( \frac{dR(t)}{dt} = \frac{R}{\sum \text{weights of active flows}} \)
    - \( \frac{dR(t)}{dt} = \frac{R}{n} \) where \( n = \# \text{active GPS flows} \) if equal \( w \).

**WFQ For Example 2**

- Suppose:
  - Equal weights: \( w(i) = 1 \), all \( i \)
  - \( t = 0 \): Pkts of size 1, 2, and 2 arrive from flows A, B, and C.
  - \( t = 4 \): Pkt of size 2 arrives from flow A.
  - \( R = 1 \) unit/sec.
- Compute the finish number of all packets.
- What is the round number when the system becomes idle?
- When does the system actually become idle?

**Round and Finish Numbers**

<table>
<thead>
<tr>
<th>N(t)</th>
<th>Round Number (Virtual Time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>A(2) arrives</td>
</tr>
<tr>
<td>3</td>
<td>Service Curve: Slope = ( \frac{R}{# \text{active flows}} )</td>
</tr>
<tr>
<td>2</td>
<td>A(1), B(1), C(1)</td>
</tr>
<tr>
<td>1</td>
<td>B(1), C(1)</td>
</tr>
</tbody>
</table>

- Actual finish times
### Weighted Fair Queueing (WFQ)

**Idea**
- Compute the time that a pkt will complete service in a GPS scheduler
- Serve pkts in order of GPS virtual finishing times
  - i.e., simulate GPS on the side and schedule accordingly

**Finish Number of pkt k, flow i**
- \( F(k) = \max \{ F(k-1), N(t) \} + \frac{L(k)}{w(i)} \)
  - Simplest case: Equal weights \( (w(i) = w(j) = 1) \)

**Round Number N(t)**
- Number of service rounds required to serve a pkt in GPS
  - Rounds are completed in variable time
    - Depends on number of active flows

**Flow i is Active if:** \( F(k) > N(t) \)
- At time \( t \), latest pkt from flow i is pkt k

### Finish Numbers And Rounds

**At \( t = 0 \)**
- A: \( F(1) = \max \{ F(0), N(0) \} + \frac{L(1)}{1} = \max \{ 0, 0 \} + 1 = 1 \)
- B, C: \( F(1) = \max \{ 0, 0 \} + \frac{2}{1} = 2 \)

**At \( t = 4 \)**
- A: \( F(2) = \max \{ F(1), N(4) \} + \frac{L(2)}{1} \)
  - Where \( N(4) = N(3) + \frac{R}{w(\text{active}+)} = 1 + \frac{1}{2} = 3/2 \)
  - \( F(2) = \max \{ 1, 3/2 \} + 2 = 7/2 \)
  - Result would be finish numbers that are 8 times higher

**Real packet scheduler transmits**
- A in [0-1], B in [1-3], C in [3-5], A in [5-7]

**F(k) and R(t) are virtual times**
- Could choose to simulate bit-by-bit (or something else)

### WFQ Implementation

**When Packet Arrives**
- Classify pkt based on src and dst IP addresses and ports
- Look up \( F(k) \) of last pkt served or waiting to be served
- Recompute round number \( N(t) \) using iterated deletion
  - \#active flows can go up by at most 1, but can go down to 1
  - Start with \#active flows at time of previous arrival
  - Simulate GPS: If a flow finishes, recompute \( N(t) \)
  - Repeat until no flows become inactive

**When Packet Finishes Service**
- Select packet with lowest finish number from the priority queue to start actual transmission

### Evaluation Of WFQ

**Desirable Properties**
- Provides flow isolation (because it approximates GPS)
- Under certain conditions, a flow can obtain a worst-case end-to-end queueing delay that is independent of the number of hops
  - Provides real-time guarantees
- Gives users an incentive to use intelligent flow control

**Difficulties**
- Requires per flow state
- Expenses iterated deletion algorithm to update round numbers
- Requires explicit sorting of output queue on service tag (priority queue)
**Self-Clocked Fair Queuing (SCFQ)**

- WFQ Finish Number
  \[ F(k) = \text{Max} \{ F(k-1), N(t) \} + \frac{L(k)}{w(flow)} \]
- SCFQ Finish Number
  \[ F(k) = \text{Max} \{ F(k-1), CF \} + \frac{L(k)}{w(flow)} \]
  - \( CF \): Finish number of packet currently in GPS service
- Speeds up round number computation
  - Use CF instead of round number when packet arrives to empty queue

**Packet Dropping**

- Dropping wastes partially completed work
  \[ \Rightarrow \text{Avoid if possible; else do it intelligently} \]
- Drop Position
  - Tail Dropping (Drop incoming pkt)
    - Default approach which is easy to implement
  - Head Dropping (Waste memory bandwidth)
    - Drop occurs sooner \( \Rightarrow \) Earlier drop detection by endsystem
- Random Dropping (Complex)
  - Distributes losses fairly among flows
  - Statistically punishes bandwidth hogs
- Entire Queue Dropping
  - Drop entire longest per-flow queue when it is full
  - Idea: Longest queue belongs to worst-behaved flow

**Early Or Overloaded Drop**

- Early Random Drop
  - Drop with probability \( p \) whenever aggregate queue length exceeds a threshold
  - Attempts to punish misbehaving flows (but really doesn’t)
- Random Early Detection (RED)
  - Drop based on exponential average of queue length
    - Drops only when overload is sustained
  - Can mark offending packets instead of dropping
    - Modified endsystems can support congestion avoidance like DECbit scheme or Explicit Congestion Notification (ECN)

<table>
<thead>
<tr>
<th>Avg Queue Length</th>
<th>Discard</th>
<th>Drop w/ prob ( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td>hi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Random Early Detection (RED)**

- RED algorithm with parameters \( lo, hi \) and \( pMax \)
  - Queue arriving pkt if \( qAvg < lo \)
  - Drop arriving pkt with probability \( P \) if \( (lo < qAvg < hi) \)
    - where \( P = \frac{pDrop}{1 - count \times pDrop} \)
    - and \( count \) = Number of packets accepted when \( lo \leq q(k) \leq hi \)
  - Drop arriving pkt if \( qAvg \geq hi \)
- \( qAvg \), average queue length, is an exponential avg
  \[ qAvg(k+1) = (1-a) \times qAvg(k) + a \times q(k) \]
RED

- RED is trying to signal TCP thru pkt drops that it should slow down
  - A single drop in a TCP flow will cause duplicate ACKs which will in turn will reduce cwnd
  - Wants to drop one pkt from each flow until out of congestion
- RED operates in tail drop mode when instantaneous queue length reaches the buffer capacity
- Selecting parameters is problematic

Active Queue Management (AQM)

Source-Based Congestion Avoidance

- Idea: Source slows down when queue builds up
  - Queue must be increasing if RTT is increasing
  - Increment cwnd every RTT until congestion detected
  - Run congestion avoidance algorithm every 2 RTTs
- Algorithm 1
  - $cwnd' = \frac{7}{8}cwnd$ if $RTT > \frac{RTT_{max} - RTT_{min}}{2}$
  - Where $cwnd'$ is the new cwnd
- Algorithm 2
  - $cwnd' = \frac{7}{8}cwnd$ if $(cwnd-cwnd_{old}) \times (RTT-RTT_{old}) > 0$
- Algorithm 3 (Detect throughput flattening)
  - $cwnd' = cwnd - 1$ if $(R'(k) - R'(k-1)) < \frac{R'(1)}{2}$
  - Where $R'(k)$ is the throughput when k pkts are inflight (window size)
    - $R'(k) = k/RTT(k)$ where $RTT(k)$ is the RTT when $cwnd = k$ pkts

TCP Vegas

- Idea
  - Expected rate $R^* = cwnd/RTT_{min}$
    - $RTT_{min}$ = RTT when there is no congestion (no queueing)
  - Calculate actual rate $R' = cwnd/RTT$
  - Operate system at slight overload (extra pkts)
    - Want $R'$ such that $a < (R^*-R') < b$ where $a$ and $b$ are parameters
- Adjustment Regions
  - Low Load: $(R^*-R') < a \Rightarrow cwnd' = cwnd + 1$
  - High Load: $(R^*-R') > b \Rightarrow cwnd' = cwnd - 1$
  - Timeout: Multiplicative decrease
- Example ($R^*=100$ msec, $L=1$ KB, $a=30$ Kbps, $b=60$ Kbps)
  - $a \times R^* = 3000$ bytes = 3 pkts
  - $b \times R^* = 6000$ bytes = 6 pkts
  - 3 to 6 extra buffers
**Guaranteed Service Rates**

- Guaranteed rates for flows 1..11
  - \( g(1) = 0.5; \ g(k) = 0.05, \ k=2..11 \)
- Equal length pkts
- In one round, serve 10 pkts from flow 1 and 1 pkt from each of the flows 2..11
  - Want: \( g(i)/g(j) = w(i)/w(j) \)

- \( w(1) = 10, \ w(k) = 1, \ k=2..11 \)

<table>
<thead>
<tr>
<th>Flow 1</th>
<th>1 2 3 4 5 6 7 8 9 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow 2</td>
<td>1 Pkt</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Flow 11</td>
<td>1 Pkt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FIFO</th>
<th>1 2 3 4 5 6 7 8 9 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFQ</td>
<td>1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

**WFQ and Guaranteed Performance**

- Provides performance guarantees
- End-to-End Delay Bound
  - Possible if flow’s burstiness (traffic surges) is limited
  - Flow should be leaky-bucket regulated
- Bandwidth Bound
  - \( g(i) = R w(i)/w(+) \) (Sum of all weights)

**Leaky Bucket Regulator/Shaper**

- Traffic Shaper
  - Smooth out flow and reduce clumping
  - If server transmits a B-byte pkt, consume B bytes of tokens
  - Long-term outgoing rate is \( \rho \)
  - Maximum burst is \( \beta \)

**Effect of Token Bucket**

- Rate = Volume / (Measurement Interval)
  - Measurement interval is large relative to event times
- Policer: Omit the data buffer
Parekh and Gallegher's Result

Queueing and Transmission Delay Bound

- Excludes propagation delay but easy to add in
- Independent of whether other flows are shaped or not
- $Q + T \leq \frac{\beta}{g} + \frac{(H-1)L}{g} + L_{\text{Max}} \left( \frac{1}{R(1)} + \ldots + \frac{1}{R(H)} \right)$