Programmable Scheduling

Lecture 18, Computer Networks (198:552)
Fall 2019
Scheduling in switch pipelines

• Packets wait in buffers/queues until serviced
• Two possibilities: Input-queued vs. output-queued
• Suppose there are pkts on port 1 to both 2 and 3
  • But suppose port 2 is clogged
  • Port 1’s packets towards port 3 should not be delayed (HOL block)
• Better to have queues represent output port contention
Why care about packet scheduling?

• Significantly influences how packets are treated regardless of the endpoint transport
  • Implementations of Quality of Service (QoS) within large networks
  • Implications for net neutrality debates

• Intellectually interesting and influential (“top 10”) question
  • Classic Demers et al paper (WFQ) has ~ 1500 citations
  • Important connections to sched literature (e.g., job scheduling)

• Scheduling algorithms influence many daily life decisions 😊
Scheduling vs. Buffer Management

• How packets enter vs. how packets leave the switch buffer
  • Typical buffer management: Tail-drop

• How should buffer memory be partitioned across ports?
  • Static partitioning?
    • Inefficient: even if port 1 has nothing to send, might drop port 2

• Also want fair sharing of buffer
  • If output port 1 is congested, why should port 2 traffic suffer?

• State of the art: dynamic buffer sharing algorithms
Fair Resource Allocation

Allocate how? among who?
Fair and efficient use of a resource

• Suppose \( n \) users share a single resource
  • Like the bandwidth on a single link
  • E.g., 3 users sharing a 30 Gbit/s link

• What is a **fair** allocation of bandwidth?
  • Suppose user demand is “elastic” (i.e., unlimited)
  • Allocate each a \( \frac{1}{n} \) share (e.g., 10 Gbit/s each)

• But **fairness is not enough**
  • Which allocation is best: [5, 5, 5] or [18, 6, 6]?
  • [5, 5, 5] is fair but [18, 6, 6] is more efficient
  • What about [5, 5, 5] vs. [22, 4, 4]?
Fair use of a single resource

• What if some users have inelastic demand?
  • E.g., 3 users where 1 user only wants 6 Gbit/s
  • And the total link capacity is 30 Gbit/s
• Should we still do an “equal” allocation?
  • E.g., [6, 6, 6]
  • But that leaves 12 Gbps unused
• Should we allocate in proportion to demand?
  • E.g., 1 user wants 6 Gbps, and 2 each want 20 Gbit/s
  • Allocate [4, 13, 13]?
• Or, give the least demanding user all she wants?
  • E.g., allocate [6, 12, 12]?
Max-min fairness

• Protect the less fortunate
  • Any attempt to *increase* the allocation of one user necessarily *decreases* the allocation of another user with equal or lower allocation

• Fully utilize a bottleneck resource
  • If demand exceeds capacity, the link is fully used
Max-min fairness for a single resource

- Progressive filling algorithm (also called waterfilling)
  - Grow all rates until some users stop having demand
  - Continue increasing all remaining rates until link is fully utilized

- If all users have elastic demands, single resource shared evenly

```
<table>
<thead>
<tr>
<th>Link rate L</th>
<th>N elastic users</th>
<th>Link rate L/N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Link rate L/N</th>
<th>Link rate L/N</th>
<th>Link rate L/N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Allocation over multiple resources

Three users A, B, and C
Two 30 Gbit/s links

• Maximum throughput: [30, 30, 0]
  • Unfair: total throughput of 60, but user C starves

• Max-min fairness: [15, 15, 15]
  • Inefficient: everyone gets equal share, but throughput is just 45

• Proportional fairness: [20, 20, 10]
  • Allocate inversely proportional to resource use per bit
  • C is penalized for using two busy links, as opposed to one
Allocate fairly among who?

- Traffic sources?
  - Web servers, video servers, etc. need more than their fair share
- Traffic destinations?
  - Vulnerable to malicious sources denying service to receivers
- Source-destination pairs?
  - Can open up connections to many destinations
- Application flows? (i.e., src + dst + transport ports)
  - Malicious app can start up many such flows
- Administrative entities? (e.g., Rutgers NetID, ISP, …)
  - How should a router identify packets belonging to an entity?

Abstract entity: a flow
Packet Scheduling Algorithms

Which packet to send next? (order)
When to send the next packet? (timing)
A taxonomy

- Granularity of allocation
  - Per-packet vs. per-flow vs bit-by-bit
- Pre-emptive vs. non-pre-emptive
  - Do you interrupt the current packet/flow if another shows up?
- Size-aware vs. unaware
  - Do you consider flow or packet sizes in scheduling?
- Class-based (strict priority) vs. shared
  - Are some flows strictly higher priority than others?
- Work-conserving vs. non-work-conserving
  - Do you always use spare link capacity when there is demand?
Examples of scheduling algorithms (1/3)

• FIFO over packets

• Round-robin over packets of different flows

• Shortest Remaining Processing Time (SRPT)
  • Flow-size-aware allocation which strictly prioritizes short flows
  • Also called shortest flow first in some contexts
  • Flow-size-unaware variant may predict demand using known flow size distribution
Examples of scheduling algorithms (2/3)

• Processor sharing
  • Assume each flow gets a fair share of the link every unit of time
  • Ideal: each flow starts receiving service immediately upon arrival

• Rate limiting
  • Non-work-conserving: flow can’t send even if more demand than limit

• Class-based strict prioritization
  • Pre-determined flow classes with strict priorities over each other
  • Starve low priority flows if higher priority flows are always sending
Examples of scheduling algorithms (3/3)

• **Hierarchical policies**
  - Arrange existing scheduling policies in a tree

• **Example:**
  - Rate-limit A + B
  - Fair-share among A and B within limit
  - Fair-share among A+B and C

• **Complex multi-tenant isolation policies**
There’s not one optimal scheduling

<table>
<thead>
<tr>
<th>Flow ID</th>
<th>Size</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>F3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F5</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Multiplexing Avoids HOL blocking

Serialization Reduces flow completion time

Source: Workload adaptive flow scheduling, by Faisal et al. CoNEXT 2018
Exercise: When does a flow finish?

• Consider a mix of “long” and “short” flows arriving at a Q
  • Ex: A flow may have as few as 2 packets or as many as $10^5$

• Suppose a scheduling algorithm provides each flow:
  • An average per-packet delay $d$ (e.g., 50 ms)
  • An average link bandwidth share $t$ (e.g., 10 Mbit/s)

• Which among $d$ & $t$ determines
  • when a short flow finishes?
  • when a long flow finishes?
Push In First Out (PIFO)

A common primitive for many scheduling algorithms
Key ideas

• Scheduling algorithms determine order and timing of packet departures from a queue

• Typically, relative order of buffered packets doesn’t change upon new packet arrivals

• Implement scheduling through a priority-queue-based data structure (PIFO)
  • Push-In: pkts have arbitrary ranks; push anywhere into queue
  • First-Out: always dequeue from the head of the queue
Programmable Scheduler

• To program the scheduler, program the rank computation

```
f = flow(pkt)
...
... 
p.rank = T[f] + p.len
```

(rank computation)
Programmable scheduling in the pipeline

- Parser
- Ingress pipeline
- Rank Computation
- Scheduler
- Egress pipeline
- Deparser

In → Rank Computation → Egress pipeline → Out
Fair queueing

Rank Computation
1. $f = \text{flow}(p)$
2. $p\text{.start} = \max(T[f]\text{.finish, virtual_time})$
3. $T[f]\text{.finish} = p\text{.start} + p\text{.len}$
4. $p\text{.rank} = p\text{.start}$
Token-bucket rate limiting

Rank Computation
1. tokens = min( tokens + rate * (now – last), burst)
2. p.send = now + max( (p.len – tokens) / rate, 0)
3. tokens = tokens - p.len
4. last = now
5. p.rank = p.send
Generalizations

• Use a **hierarchy of PIFOs** to implement hierarchical policies

• Use a **shaping PIFO** to implement non-work-conserving scheduling policies
Implementation: Challenges

• Packet buffer is big! Schedule among all those packets?
  • Maintaining a sorted list of 64K packets?
  • Instead, make flow-level scheduling decisions
  • With FIFO order among packets of a given flow

• Sorting even just flows at line rate
  • Line-rate insertion and removal from hardware priority queue with
    1000s of flow elements
  • Use fast flip-flops and pipelined logic
Implementation: Flow-level PIFOs
Encoding HPFQ in a PIFO mesh
Fair Queueing

ACM SIGCOMM '89

Alan Demers, Srinivasan Keshav, and Scott Shenker
An ideal to emulate: Processor sharing

• Fair-share bandwidth in the most fine-grained fashion possible
  • If there are N active flows, each flow gets \( \frac{1}{N} \) of the link rate
  • “Bit by bit round robin” (BR)

• Implementing BR directly on routers is unrealistic. Why?
  • One reason: consider the processing of the bit downstream
  • E.g., where to route the bit?
Emulate bit-by-bit round robin (BR)?

• How about round robin over packets?

• Unfair! A flow can use larger packets and gain larger bandwidth

• Instead, determine when a packet would finish with BR
  • Depends only on packet arrival time & # of active flows
  • Let’s call this the “virtual finish time”

• FQ: Transmit packets in the order of the virtual finish times
  • Buffer management: drop pkt of the flow with the largest backlog
Deficit Round Robin

- Router-friendly implementation of a WFQ scheme