Universal Packet Scheduling

Radhika Mittal, Rachit Agarwal,
Sylvia Ratnasamy, Scott Shenker

UC Berkeley
Many Scheduling Algorithms

• Many different algorithms
  – FIFO, FQ, virtual clocks, priorities…

• Many different goals
  – fairness, small packet delay, small FCT…

• Many different contexts
  – WAN, datacenters, cellular…
Many Scheduling Algorithms

• Implemented in router hardware.

• How do we support different scheduling algorithms for different requirements?
  – Option 1: Change router hardware for each new algorithm
  – Option 2: Implement all scheduling algorithms in hardware
  – Option 3: Programmable scheduling hardware*

*Towards Programmable Packet Scheduling, Sivaraman et. al., HotNets 2015
Many Scheduling Algorithms

- Implemented in *router hardware*.

- How do we support *different scheduling algorithms* for different requirements?
  - Option 1: Change router hardware for each new algorithm
  - Option 2: Implement *all* scheduling algorithms in hardware
  - Option 3: Programmable scheduling hardware*

*Towards Programmable Packet Scheduling, Sivaraman et. al., HotNets 2015*
Many Scheduling Algorithms

- Implemented in router hardware.

- How do we support different scheduling algorithms for different requirements?
  - Option 1: Change router hardware for each new algorithm
  - Option 2: Implement all scheduling algorithms in hardware
  - Option 3: Programmable scheduling hardware*

*Towards Programmable Packet Scheduling, Sivaraman et. al., HotNets 2015
Many Scheduling Algorithms

- Implemented in router hardware.

- **How do we support different scheduling algorithms for different requirements?**
  - Option 1: Change router hardware for each new algorithm
  - Option 2: Implement all scheduling algorithms in hardware
  - Option 3: Programmable scheduling hardware*

*Towards Programmable Packet Scheduling, Sivaraman et. al., HotNets 2015
We are asking a new question.....

How do we support different scheduling algorithms for different requirements?

Is there a *universal* packet scheduling algorithm?
UPS: Universal Packet Scheduling Algorithm

A single scheduling algorithm that can imitate the network-wide output produced by any other algorithm.
How can a single algorithm imitate all others?
Network Model

Input Traffic → CORE NETWORK

INGRESS
Network Model

Input Traffic → Scheduling Algorithm

INGRESS → CORE NETWORK
Network Model

Input Traffic

(Optional) Header Initialization

(INGRESS)

Scheduling Algorithm

CORE NETWORK

Output Traffic

(EGRESS)
Network Model

Input Traffic

(Optional) Header Initialization

INGRESS

Scheduling Algorithm

CORE NETWORK

Output Traffic tied to Scheduling Algorithm

Output Traffic

EGRESS
Network Model

Goal: Minimize Mean FCT

Input Traffic

Priority Value
Flow Size

INGRESS

Priority Scheduling

CORE NETWORK

Output Traffic
EGRESS
Network Model

Goal: Fairness

Input Traffic → CORE NETWORK → Output Traffic

INGRESS

EGRESS

FQ
Network Model

Goal: Weighted Fairness

Input Traffic

Flow Weights

INGRESS

WFQ

CORE NETWORK

Output Traffic

EGRESS
Network Model

* Uses packet header state to make scheduling decisions

* Output Traffic tied to Header Initialization

INGRESS

Input Traffic

Header Initialization

CORE NETWORK

Scheduling Algorithm*

Output Traffic

EGRESS
Network Model

Input Traffic

Header Initialization

INGRESS

UPS?

CORE NETWORK

Output Traffic

EGRESS
How do we formally define and evaluate a UPS?
Defining a UPS

**Theoretical Viewpoint:**
Can it replay a given schedule?

**Practical Viewpoint:**
Can it achieve a given objective?
Theoretical Viewpoint

Can it replay a given schedule?
Only requirement from original schedule:
Output Times are viable

Input Traffic

(Optional) Header Initialization

INGRESS

Arbitrary Scheduling Algorithm

CORE NETWORK

Output Times $o(p)$ for a packet $p$

EGRESS
Replaying the Schedule, given $o(p)$

Input Traffic

Header Initialization

INGRESS

CORE NETWORK

UPS

Output Times $o'(p)$ for a packet $p$

EGRESS

For every packet $p$, $o'(p) \leq o(p)$
Pragmatic Constraints on a UPS

Obliviousness: For initializing p’s header, use only o(p) and path(p)
**Obliviousness:** For initializing p’s header, use only $o(p)$ and path(p)
Pragmatic Constraints on a UPS

Input Traffic → Header Initialization

INGRESS → UPS → CORE NETWORK → EGRESS

Output Times $o'(p)$ for a packet $p$

Obliviousness: For initializing $p$'s header, use only $o(p)$ and $\text{path}(p)$
Pragmatic Constraints on a UPS

Obliviousness: For initializing p’s header, use only o(p) and path(p)
Pragmatic Constraints on a UPS

Obliviousness: For initializing p’s header, use only o(p) and path(p)

Limited State: Scheduling can use only header state and static information

Input Traffic

Header Initialization

INGRESS

UPS

CORE NETWORK

Output Times

o’(p) for a packet p

EGRESS
Pragmatic Constraints on a UPS

**Limited State:** Scheduling can use only header state and static information

**Obliviousness:** For initializing p’s header, use only \( o(p) \) and path(p)

**Input Traffic**

**INGRESS**

**Output Times**

\( o'(p) \) for a packet p

**CORE NETWORK**

**EGRESS**

**Header Initialization**
We call this Blackbox Initialization

**Limited State:** Scheduling can use only header state and static information

**Obliviousness:** For initializing p’s header, use only o(p) and path(p)

Input Traffic

Header Initialization

INGRESS

CORE NETWORK

Output Times o’(p) for a packet p

EGRESS
Basic Existence and Non-existence Results

There exists a UPS under *Omniscient Initialization* when scheduling time at every hop is known

No UPS exists under *Blackbox Initialization* when only the final output time is known

See paper for proofs.
How close can we get to a UPS?
Key Result: Depends on congestion points

<table>
<thead>
<tr>
<th>No. of Congestion Points per Packet</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>✗</td>
</tr>
</tbody>
</table>

See paper for proofs.
Can we achieve this upper bound?
Can we achieve this upper bound?

Yes, LSTF!
Least Slack Time First

- Packet header initialized with a slack value
  - slack = maximum tolerable queuing delay

- At the routers
  - Schedule packet with least slack time first
  - Update the slack by subtracting the wait time
## Key Results

<table>
<thead>
<tr>
<th>No. of Congestion Points per Packet</th>
<th>General</th>
<th>LSTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

See paper for proofs.
## Not all algorithms achieve upper bound

<table>
<thead>
<tr>
<th>No. of Congestion Points per Packet</th>
<th>General</th>
<th>LSTF</th>
<th>Priorities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>3</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

See paper for proofs.
How well does LSTF perform empirically?
Empirically, LSTF is (almost) universal

- ns-2 simulation results on realistic network settings
  - Less than 3% packets missed their output times
  - Less than 0.1% packets are late by more than one transmission time
Summarizing the theoretical viewpoint

• Evaluate the ability to replay a schedule, given its final output times

• Analytical Results:
  – No UPS exists
  – LSTF comes as close to a UPS as possible

• Empirical Results: LSTF is *almost* universal!
Practical Viewpoint

Can it achieve a given objective?
Achieving various network objectives

- Slack assignment based on heuristics
- Comparison with state-of-the-art
- Three objective functions
  - Tail packet delays
  - Mean Flow Completion Time
  - Fairness
Tail Packet Delays

Slack Assignment: Same slack for all packets

State-of-the-art: FIFO, FIFO+

Results:
• Identical to FIFO+.
• Smaller tail packet delays compared to FIFO.
Mean Flow Completion Time

Slack Assignment: Proportional to flow size

State-of-the-art: SJF, SRPT

Results:
• Mean FCTs comparable to both SJF and SRPT.
Fairness

Slack Assignment: Inspired by Virtual Clocks

\[ \text{slack}(p_0) = 0 \]
\[ \text{slack}(p_i) = \max(0, \text{slack}(p_{i-1}) + \frac{1}{r_{\text{est}}} - (i(p_i) - i(p_{i-1}))) \]
\[ r_{\text{est}} = \text{Estimate of fair share rate} \]

State-of-the-art: Fair Queuing (FQ)

Results:

- Eventual convergence to fairness for long-lived flows.
- FCTs roughly comparable to FQ for short-lived flows.
  - Higher sensitivity to fair share rate estimate \((r_{\text{est}})\)
Active Queue Management (AQM)

- Routers send feedback in the form of dropping or marking appropriate packets.

- LSTF facilitates AQM from the edge:
  - It does not matter where the packets are dropped or marked.
  - *Used slack* value can be used for deciding which packets are to be dropped or marked.

- Performs comparable to FQ-CoDel and DCTCP (ECN).
Results Summary

• Theoretical results show that
  – There is no UPS under blackbox initialization
  – LSTF comes as close to a UPS as possible
  – Empirically, LSTF is very close

• LSTF can be used in practice to achieve a variety of network-wide objectives.
Implication

- Less need for many different scheduling and queue management algorithms.
- Can just use LSTF, with varying initializations.
There are still a bunch of open questions!
Open Questions

What is the least amount of information needed to achieve universality?

Are there tractable bounds for the degree of lateness with LSTF?

How do we achieve multiple objectives simultaneously?

What is the class of objectives that can be achieved with LSTF in practice?
Conclusion

• Theoretical results show that
  – There is no UPS under blackbox initialization.
  – LSTF comes as close to a UPS as possible.
  – Empirically, LSTF is very close.
• LSTF can be used in practice to achieve a variety of network-wide objectives.

Contact: radhika@eecs.berkeley.edu

Code: http://netsys.github.io/ups/

Thank You!