Fast, Scalable, and Programmable Packet Scheduler in Hardware

Vishal Shrivastav
Cornell University
Packet Scheduling 101

Packet Scheduler
*focus of this work*

Scheduling Algorithm
specifies **when** and **what order** to schedule packets onto the wire

Packet Queues
output interface

To Wire

**Packet Queues**

**Packet Scheduling Algorithm**

specifies **when** and **what order** to schedule packets onto the wire

**Packet Scheduler**
*focus of this work*

**Packet Queues**

**output interface**

**To Wire**

**Packet Scheduler**
*focus of this work*

**Packet Queues**

**output interface**

**To Wire**
Desirable Properties of a Packet Scheduler

Programmability
- Express wide-range of packet scheduling algorithms

Scalability
- Scale to 10s of thousands of flows
  [SENIC - NSDI’14] [Carousel - SIGCOMM’17]

Performance
- Link speed
- Time budget for scheduling decisions
  e.g., 120 ns for MTU pkt @ 100Gbps
- New transport protocols
  e.g., Fastpass, Ethernet TDMA
- Circuit-Switched network designs
  e.g., Shoal, RotorNet
- Transmit packets at precise times
  e.g., at ns-precision in Shoal

Make scheduling decisions within deterministic 10s of nanoseconds
Desirable Properties of a Packet Scheduler

Challenging to achieve all three properties (programmability, scalability, and performance) simultaneously
# State-of-the-art Packet Schedulers

## Generality vs. Performance

<table>
<thead>
<tr>
<th></th>
<th>Programmability</th>
<th>Scalability</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Hardware</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. FIFO</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2. PIFO, UPS (priority queue abstraction)</td>
<td>✓ *</td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>

*has some limitations*
Can we design a packet scheduler that is simultaneously programmable, scalable, and high performance?

We present PIEO (Push-In-Extract-Out) scheduler in hardware

- **Programmable**
  - more expressive than any state-of-the-art hardware packet scheduler

- **Scalable**
  - easily scales to 10s of thousands of flows

- **High Performance**
  - makes scheduling decisions in $O(1)$ time [4 clock cycles]
Can we design a packet scheduler that is simultaneously programmable, scalable, and high performance?

We present PIEO (Push-In-Extract-Extract-Out) scheduler in hardware.

**Programmable**
more expressive than any state-of-the-art hardware packet scheduler

**Scalable**
easily scales to 10s of thousands of flows

**High Performance**
makes scheduling decisions in $O(1)$ time [4 clock cycles]
PIE O Scheduling Abstraction

when an element becomes eligible for scheduling?
encode using a $t_{\text{eligible}}$ value

what order to schedule amongst eligible elements?
encode using a rank value

whenever the link is idle:
among all elements satisfying the eligibility predicate $t_{\text{current}} \geq t_{\text{eligible}}$
schedule the smallest ranked element

PIE O Abstraction — “schedule the smallest ranked eligible element”
strictly more expressive than a priority queue abstraction, e.g., PIFO, UPS
**Push-In-Extract-Out Primitive**

- **enqueue(18, 1)**  
  “Push-In”  
  inserts element at position dictated by its rank value

- **dequeue()**  
  “Extract-Out”  
  returns “smallest ranked eligible” element

- **dequeue()**  
  returns a specific element

- The **element** is programmed based on the choice of scheduling algorithm.

- The **ordered list** increases with the rank value.

- **enqueue(18, 1)** results in positions: 10, 12, 13, 16, 19, 21, 22.
  The current rank value is 10, and it increases to 12, 13, 16, 19, 21, 22.

- For dequeue(), the **filter** is: \( t_{current} \geq t_{eligible} \).
Expressiveness of PIEO

- **Work conserving**
  - e.g., DRR, WFQ, WF$^2$Q
- **Non-work conserving**
  - e.g., Token Bucket, RCSP
- **Hierarchical scheduling**
  - e.g., HPFQ
- **Asynchronous scheduling**
  - e.g., Starvation avoidance, D$^3$
- **Priority scheduling**
  - e.g., SJF, SRTF, LSTF, EDF
- **Complex scheduling policies**
  - mixture of shaping and ordering

---

for each element:

- calculate `start_time` and `finish_time`
- at time `x`, all elements s.t. `virtual_time(x) >= start_time`:
  - schedule element with smallest `finish_time`

---

programming PIEO

- `rank = finish_time`
- `t_{eligible} = start_time`

Predicate for filtering at dequeue at time `x`:

```
(virtual_time(x) >= t_{eligible})
```
Can we design a packet scheduler that is simultaneously programmable, scalable, and high performance?

We present PIEO (Push-In-Extract-Out) scheduler in hardware.

**Programmable**
more expressive than any state-of-the-art hardware packet scheduler

**Scalable**
easily scales to 10s of thousands of flows

**High Performance**
makes scheduling decisions in O(1) time [4 clock cycles]
PIEO primitive relies on an **ordered list** datastructure.
Is it fundamentally necessary to access and compare $O(N)$ elements in parallel to maintain an (exact) ordered list (of size $N$) in $O(1)$ time?

We present a design that can maintain an (exact) ordered list in $O(1)$ time, but only needs to access and compare $O(\sqrt{N})$ elements in parallel.

**Key Insight**

“All problems in computer science can be solved by another level of *indirection*”

—David Wheeler
Hardware Architecture

Q: How to zoom into the correct sublist in O(1) time?
Q: How to read/update/write an entire sublist in O(1) time?
Q: How to filter + extract-min in O(1) time?
Q: What to do when enqueue into a full sublist?

Detailed answers in the paper !!!

enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

..... at the cost of 2X memory overhead
Implementation

- Implemented PIEO scheduler on a Stratix V FPGA
  - 234K logic modules (ALMs)
  - 6.5MB SRAM
  - 40Gbps interface bandwidth

- ~1300 LOCs in System Verilog
4 cycles per primitive op, i.e., 50ns @ 80MHz
- pipelining
- ASIC target, e.g., 4ns @ 1GHz

But not as fast as PIFO — 1 cycle per primitive op
Beyond Packet Scheduling

- PIEO as an O(1)-time generic Priority Queue

- PIEO as an Abstract Dictionary Data Type
  - act as a (key, value) store, indexed by keys
  - search, insert, delete, and update in O(1) time
  - efficiently do complex ops like range filtering over keys
  - ..... while also being reasonably scalable

PIEO as a key basic building block in the era of hardware-accelerated computing
Conclusion

Two Key Contributions:

• A new programmable abstraction and primitive for packet scheduling
  - more expressive than any state-of-the-art hardware packet scheduler

• A fast and scalable hardware design of the scheduler
  - makes scheduling decisions in 4 clock cycles
  - easily scales to 10s of thousands of flows

Scalability

FIFO (Hardware)

PIEO (Hardware)

High Performance
FPGA code for the implementation of PIEO scheduler is available at: https://github.com/vishal1303/PIEO-Scheduler

Email: vishal@cs.cornell.edu
Webpage: http://www.cs.cornell.edu/~vishal/

Thank you!