Slytherin: Dynamic, Network-assisted Prioritization of Tail Packets in Datacenter Networks

Hamed Rezaei, Mojtaba Malekpourshahraki, Balajee Vamanan
Diverse datacenter applications

• Datacenters host *diverse* applications

  • Foreground applications
    • Respond to user queries (e.g., Web search)
    • Short flows (e.g., < 32 KB)

  • Background applications
    • Update data for foreground applications (e.g., Web crawler)
    • Long flows

Substantial diversity in datacenter applications
Multiple Network Objectives

- **Requirements**
  - *Foreground* applications
    - Sensitive to tail (e.g., 99\textsuperscript{th} %-ile) flow completion times
  - *Background* applications
    - Sensitive to throughput

Datacenter network must provide *both* low tail flow completion times for short flows *and* high throughput for long flows
Existing state-of-the-art

- **Load balancing**
  - Presto [SIGCOMM ’15], Hula [SOSR ’16]
  - Conga [SIGCOMM ’14]
  - Problem exists even with perfect load balancing

- **Congestion control**
  - DCTCP [SIGCOMM ’10], D2TCP [SIGCOMM ’12]
  - Does not specifically target short flows (tail latency)

- **Packet scheduling**
  - Information-aware
    - pFabric [SIGCOMM ’13]
    - PASE [SIGCOMM ’14]
    - Apriori knowledge of flow sizes
  - Information-agnostic
    - PIAS [NSDI ’15]
    - Most relevant to our proposal

Packet scheduling plays a direct role in the flow completion times of short flows (our focus on this paper)
Shortest Job First (SJF)

- PIAS aims to implement SJF Practically
  - Improves FCT

- SJF per switch $\neq$ SJF for network
  - Optimal for a single switch (PIAS [NSDI ‘15])
  - NOT optimal for the whole network (pFabric [Sigcomm ‘13])

PIAS (SJF per switch) is locally but not globally optimal
Key question!

• Is it possible to *identify* and *prioritize tail packets*, to improve the *tail flow completion times beyond PIAS*?
Our contributions

• **Key insight**: Tail packets often incur congestion at multiple points in the network

• Slytherin
  • **Step I**: Targets tail packets
    • Uses ECN marks to identify tail packets
  • **Step II**: Prioritizes tail packets
    • Uses multi-level queues

• Drastically reduces network queuing
  • 2x lower queue length

Slytherin achieves 19% lower tail flow completion times without losing throughput
Outline

• Datacenter network objectives
• Existing proposals
• Our contributions
  • Idea and Opportunity
  • Slytherin Design
  • Methodology
  • Results
• Closing remarks
Idea and opportunity

• High-level idea
  • Packets that incur congestion at multiple points
    • Likely to fall in the tail
    • Worsen tail FCT

Maximum time slot = 2

Num of slots required before dequeue

S1 S2 S3 S4 S5 S6
Idea and opportunity

• High-level idea
  • Packets that incur congestion at multiple points
    • Likely to fall in the tail
    • Worsen tail FCT

Num of slots required before dequeue

Maximum time slot =

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Idea and opportunity (cont.)

• Could packets marked at multiple points, influence tail?
  • What is the fraction of packets that
    • are marked at more than one switch **AND**
    • fall in the tail

• Experimental analysis

Substantial opportunity at high loads!
Outline

• Datacenter network objectives
• Existing proposals
• Our contributions
• Idea and Opportunity
  • Slytherin Design
• Methodology
• Results
• Closing remarks
## High level overview

<table>
<thead>
<tr>
<th>Policy intent</th>
<th>Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step I</strong></td>
<td></td>
</tr>
<tr>
<td>Identify packets that fall in the tail</td>
<td>Leverages ECN mark to identify tail packets</td>
</tr>
<tr>
<td><strong>Step II</strong></td>
<td></td>
</tr>
<tr>
<td>Prioritizes tail packets</td>
<td>Multi-level queue</td>
</tr>
</tbody>
</table>
Slytherin: Design

- Slytherin has two steps:
  - Step I: **Identify** packets that previously incur congestion
    - Use ECN marked packets
Slytherin: Design

- Slytherin has two steps:
  - Step I: *Identify* packets that previously incur congestion
  - Use ECN marked packets

- This packet previously incurred congestion
- Mark packet
Slytherin: Design

- Slytherin has two steps:
  - Step II: *Prioritize* previously congested packets
    - Use Multi-level queue
    - Supported in today’s datacenter switches
Slytherin: Design

- Slytherin has two steps:
  - Step II: *Prioritize* previously contented packets
  - Use Multi-level queue for all switches
Slytherin: Design

• Slytherin has two steps:
  • Step II: *Prioritize* previously contented packets
  • Use Multi-level queue
Slytherin: Design

• Slytherin has two steps:
  • Step II: *Prioritize* previously contented packets
  • Use Multi-level queue
Design Considerations

• Ack prioritization
  • Ack packets have the highest priority
  • Promote all Ack packets to the queue with higher priority

• Implementation
  • Most of the switches support multi-level queue
  • Implement Slytherin with some switch rules
Outline

• Datacenter network objectives
• Existing proposals
• Our contributions
• Idea and Opportunity
• Slytherin Design

• **Methodology**
• Results
• Closing remarks
Simulation Methodology

- Network simulator (ns-3 v3.28)
  - Transport protocol
    - DCTCP as underlying congestion control
  - ECN threshold
    - Threshold 25% of total queue size
  - Workloads
    - Short flows (8 - 32 KB) (web search workload)
    - long flows (1 MB) (data mining)
  - Schemes compared
    - PIAS [NSDI ‘15]
    - DCTCP [SIGCOMM ’10]
Topology

- Two level, leaf-spine topology
- 400 hosts through 20 leaf switches
- Connected to 10 spine switches in full mesh
Slytherin achieves 18.6% lower 99\textsuperscript{th} percentile flow completion times for short flows.
Average throughput (long flows)

Slytherin achieves an average increase in long flows throughput by about 32%.
CDF of queue length in switches

Slytherin significantly reduces 99th percentile queue length in switches by about a factor of 2x on average.
Conclusion

• Prior approaches
  • Leverage SJF to minimize average flow completion times

• We proposed Slytherin
  • Identifies tail packets and prioritizes them in the network switches
  • Optimizes tail flow completion times
    • Tail flow completion time → most appropriate for many applications

• Future work:
  • Improve Slytherin’s accuracy in identifying tail packets
  • Efficient switch hardware implementations

As data continues to grow at a rapid rate, schemes such as Slytherin that minimize network tail latency will become important
Thanks for the attention

mmalek3@uic.edu
Reordering

- PIAS with limited buffer for each port
- Reordering PIAS/Slytherin
Sensitivity to threshold

Graph showing the tail (99th percentile) FCT (ms) for different loads and percentages:
- Slytherin (12.5%)
- Slytherin (25%)
- Slytherin (37.5%)

Load:
- 20%
- 40%
- 60%
- 80%
Sensitivity to incast
Slytherin: Design - ACK

- Prioritize ACK packets
  - Packets will follow Slytherin rules
Tail flow completion time

• **Scenario I:** Assume we have 5 short flows as follow

<table>
<thead>
<tr>
<th>Flow ID</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1ms</td>
</tr>
<tr>
<td>20</td>
<td><strong>5ms</strong></td>
</tr>
<tr>
<td>30</td>
<td>2ms</td>
</tr>
<tr>
<td>40</td>
<td>2ms</td>
</tr>
<tr>
<td>50</td>
<td>4ms</td>
</tr>
</tbody>
</table>

Average FCT = \( \frac{(1+2+2+5+4)}{5} \)

Tail FCT = 5

• **Scenario II:**
  - which is better?

<table>
<thead>
<tr>
<th>Flow ID</th>
<th>Duration</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>