FlowValve: Packet Scheduling Offloaded on NP-based SmartNICs

ICDCS 2022
Shaoke Xi, Fuliang Li, XingWei Wang

Northeastern University
Zhejiang University
FlowValve is a parallel packet scheduler for Network Processor (NP)-based SmartNICs that offloads critical network functions of Linux TC, including classifying and scheduling.

- What are the requirements of end-host scheduling?
- The offloading idea
- Why using NP-based SmartNICs?
What are the requirements of packet scheduling?
End-host enforces complex network policies to meet SLAs for applications and tenants.

**Policy 1:** Host.Controller -> highest priority
End-host enforces complex network policies to meet SLAs for applications and tenants.

Policy 1: Host.Controller -> highest priority
Policy 2: VM1:VM2 -> weighted sharing
End-host enforces complex network policies to meet SLAs for applications and tenants.

Policy 1: Host.Controller -> highest priority
Policy 2: VM1:VM2 -> weighted sharing
Policy 3.1: VM1.KVS prior than VM1.ML
Policy 3.2: VM1.ML guaranteed 2Gbps
End-host enforces complex network policies to meet SLAs for applications and tenants.

**Policy 1:** Host.Controller -> highest priority

**Policy 2:** VM1:VM2 -> weighted sharing

**Policy 3.1:** VM1.KVS prior than VM1.ML

**Policy 3.2:** VM1.ML guaranteed 2Gbps

🌟 Fine-grained traffic control
🌟 Complex network policies
🌟 Flexible support of new algorithms
Efficient Enforcement

Single core scheduler works fine under low packet rate.

Scheduling accuracy drops under heavy traffic workloads (e.g., >10Gbps).

Root cause: Single-core scheduling hits the performance bottleneck.

Single core scheduler can easily maintain consistent queue status.

Multi-core scheduling needs inter-core coordination, which is challenging.
The Offloading Idea

Utilize multi-core hardware to accelerate packet scheduling.
Offload classifying and scheduling functions to save CPU cores.
Why using Network Processor-based SmartNICs?
Parallel Packet Processing

- **Parallel Scheduling**

Embed scheduling function in each core’s processing routine.

Many worker cores coordinate to perform scheduling algorithms.
Parallel Packet Processing

- **Parallel Scheduling**
  Embed scheduling function in each core’s processing routine.
  Many worker cores coordinate to perform scheduling algorithms.

- **Flexible Development**
  Develop new algorithms in a software manner.
Hardware Acceleration

- **Efficient Flowcache**
  Specialised cache mechanism accelerate packet classification.
  Large on-chip memory caches millions of flows.
Hardware Acceleration

- **Efficient Flowcache**
  Specialised cache mechanism accelerate packet classification.
  Large on-chip memory caches millions of flows.

- **Atomic Instruction**
  Memory engines conduct atomic arithmetic operations to alleviate multi-core locking overhead.
Virtualization Support

- **Fast Speed**
  Deliver high performance to VMs through bypassing the host networking stack.
  Meanwhile, conducting network policies on the NIC dataplane.
Challenges
Challenges

- Multi-core parallelism

How to reduce inter-core collaboration?
Challenges

● Multi-core parallelism
How to reduce inter-core collaboration?

● Constrained buffer management
How to avoid congestion on egress by handling packets on their way into TX buffers?
Challenges

● Multi-core parallelism

How to reduce inter-core collaboration?

● Constrained buffer management

How to avoid congestion on egress by handling packets on their way into TX buffers?

Insight: Abstract TX buffers as a FIFO queue and perform specialized tail drop to mix the FIFO queue with expected flow proportions.
FlowValve

- Express network policies as a scheduling tree
  - Traffic classes represent by tree nodes.
  - Flow QoS settings represent by tree paths.
FlowValve

- Express network policies as a scheduling tree
  - Traffic classes represent by tree nodes.
  - Flow QoS settings represent by tree paths.

- **Parallelly update traffic classes on multi-core NPs**
  - Estimate instant flow rate at interior nodes.
  - Enforce rate control at leaf nodes.
FlowValve

- Express network policies as a scheduling tree
  - Traffic classes represent by tree nodes.
  - Flow QoS settings represent by tree paths.
- **Parallelly update traffic classes on multi-core NPs**
  - Estimate instant flow rate at interior nodes.
  - Enforce rate control at leaf nodes.

![Diagram of FlowValve](image)
FlowValve

- Express network policies as a scheduling tree
  - Traffic classes represent by tree nodes.
  - Flow QoS settings represent by tree paths.

- **Parallelly update traffic classes on multi-core NPs**
  - Estimate instant flow rate at interior nodes.
  - Enforce rate control at leaf nodes.
Workflow

- **Frontend**
  Take in network policies to construct the scheduling tree.
Workflow

- Frontend

Take in network policies to construct the scheduling tree.

Populate tree parameters to NICs.
Workflow

- **Frontend**
  Take in network policies to construct the scheduling tree.
  Populate tree parameters to NICs.

- **Backend**
  Execute labeling function to get QoS setting labels.
Workflow

- **Frontend**
  Take in network policies to construct the scheduling tree.
  Populate tree parameters to NICs.

- **Backend**
  Execute labeling function to get QoS setting labels.
  Execute scheduling function to update traffic classes.
Scheduling Function

- Class update without synchronization on NPs is inaccurate ✗

\[ R_{ceil} = B \]

Prior: 1

Prior: 2

\[ S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \]

\[ \text{Invalid} \]

\[ \begin{array}{ccc}
S_0 & \text{NC} & \text{ML} \\
S_0 & S_1 & S_2 \\
S_0 & \text{NC} & \\
\end{array} \]

Pkt NC₁ (core 1)
Pkt ML₁ (core 2)
Pkt NC₂ (core 3)

\[ T_1 \rightarrow T_2 \]

Time
Scheduling Function

- Class update without synchronization on NPs is inaccurate
- **Sequential update leads to extreme low throughput**
Scheduling Function

- Class update without synchronization on NPs is inaccurate ✗
- Sequential update leads to extreme low throughput ✗
- Locking at the class level balances accuracy and efficiency ✓
Hierarchical Rate-limiting

- Restrict flow rate with token buckets

\[ \theta_C = \frac{b_C}{f} \]  

(token rate of class C, bit rate of class C, worker core frequency)
Hierarchical Rate-limiting

- **Restrict flow rate with token buckets**

  \[ \theta_C = \frac{b_C}{f} \]  

  (1)

- **Adjust token rate at runtime**
  
  - **Priority**
    
    \[ \theta_{S_1} = \theta_{S_0} - \Gamma_{NC}, \Gamma_{NC} = \frac{\sum L_P}{\Delta T}, P \in NC \]  

    (2)
Hierarchical Rate-limiting

- Restrict flow rate with token buckets
  \[ \theta_C = \frac{b_C}{f} \]  
  (1)

- Adjust token rate at runtime
  - Priority
    \[ \theta_{S_1} = \theta_{S_0} - \Gamma_{NC}, \Gamma_{NC} = \frac{\sum L_P}{\Delta T}, P \in NC \]  
    (2)

- Bit rate of class C
- Worker core frequency
- Token rate of class C
- Token consumption rate of class NC
- Tokens consumed by forwarded packets
- Update interval
Hierarchical Rate-limiting

- **Restrict flow rate with token buckets**
  \[ \theta_C = \frac{b_C}{f} \] (1)

- **Adjust token rate at runtime**
  - **Priority**
    \[ \theta_{S1} = \theta_{S0} - \Gamma_{NC}, \Gamma_{NC} = \frac{\sum L_P}{\Delta T}, P \in NC \] (2)
  - **Weight**
    \[ \theta_{WS} = \theta_{S1} \times \frac{1}{3}, \theta_{S2} = \theta_{S1} \times \frac{2}{3} \] (3)

- **Update interval**
- **Tokens consumed by forwarded packets**
- **Priority rate of class NC**
- **Weight rate of class NC**
- **Update interval**
- **Bit rate of class C**
- **Worker core frequency**
- **Token rate of class C**
Bandwidth Sharing

- Sharing of unconsumed tokens

\[ \theta_{lendable} = \theta_C - \Gamma_C \]

\( \text{token rate of class } C \)

\( \text{token consumption rate of class } C \)
Bandwidth Sharing

- Sharing of unconsumed tokens

\[ \theta_{lendable} = \theta_C - \Gamma_C \]

- Preferent sharing among interior classes

Example: KVS is idle. WS and ML are hungry.

WS borrows from S_2. ML borrows from KVS.
Error Analysis

- Single class rate-limiting is accurate
- **Main error: propagation delay of token rate adjustment**
Experiment Setup

● Implementation
  ○ Frontend: Python
  ○ Backend: Netronome Agilio CX 40GbE SmartNIC

● Testbed
  ○ Hardware
    ■ Netronome SmartNIC: send + schedule
    ■ Intel X710 40GbE NIC: receive
  ○ FlowValve: DPDK driver + mTCP stack
  ○ Software scheduler
    ■ Linux HTB: iperf3 traffic generator
    ■ DPDK QoS Scheduler
Evaluation

- **QoS Policy Enforcement**

Q1: Can FlowValve enforce network policies?

Q2: Can FlowValve drive line rate?
QoS Policy Enforcement

FlowValve offers better rate conformance than HTB on a 10Gbps link.

Less strict priority  Time (s)  Wrong weights

HTB

FlowValve
QoS Policy Enforcement

FlowValve drives to line rate while accurately scheduling traffic.
Evaluation

● QoS Policy Enforcement

Q1: Can FlowValve enforce network policies?
Q2: Can FlowValve drive line rate?

● Offloading Effectiveness

Q3: How many CPU cores can FlowValve save?
Q4: How does FlowValve impact transmission delay?
Offloading Effectiveness

FlowValve contributes to save at least 2 CPU cores when driving line rate.

<table>
<thead>
<tr>
<th>Packet Size (Byte)</th>
<th>FlowValve</th>
<th>DPDK QoS Scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Maximum Throughput (Mpps)</td>
<td>Maximum Throughput (Mpps)</td>
</tr>
<tr>
<td>1518</td>
<td>3.23</td>
<td>2.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.24</td>
</tr>
<tr>
<td>1024</td>
<td>4.75</td>
<td>4.49</td>
</tr>
<tr>
<td>64</td>
<td>19.69</td>
<td>9.06</td>
</tr>
</tbody>
</table>
Offloading Effectiveness

FlowValve significantly lowers delay variation.

<table>
<thead>
<tr>
<th>Bandwidth (Gbps)</th>
<th>Scheduler</th>
<th>One-way Delay (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mean</td>
</tr>
<tr>
<td>10</td>
<td>HTB</td>
<td>36.74</td>
</tr>
<tr>
<td></td>
<td>FlowValve</td>
<td>30.05</td>
</tr>
<tr>
<td></td>
<td>DPDK QoS</td>
<td>50.51</td>
</tr>
<tr>
<td>40</td>
<td>FlowValve</td>
<td>162.93 (161.01)</td>
</tr>
<tr>
<td></td>
<td>DPDK QoS</td>
<td>70.38</td>
</tr>
</tbody>
</table>
Conclusion

- FlowValve is the first parallel packet scheduler for NP-based SmartNICs that offloads critical functions of Linux traffic control.
- FlowValve offers high throughput and substantially reduces CPU burdens.
Thanks!