NetVRM: Virtual Register Memory for Programmable Networks

Hang Zhu
Tao Wang, Yi Hong, Dan R. K. Ports, Anirudh Sivaraman, Xin Jin
Data plane objects

Stateless (lifespan <= 1 packet)
- Metadata, packet headers

Stateful (lifespan > 1 packet)
- Tables, counters, meters, registers

Registers enable a new class of reg-stateful applications
The case of dynamic allocation for register memory

- **Necessity**
  - Limited register memory (e.g., a few Mb/stage)
  - Concurrent reg-stateful applications

- **Potential benefits**
  - Diminishing return
Diminishing return

NetCache with different skew workload

(a) Heavy hitter detection.  (b) Newly opened TCP connections.  (c) Superspreader detection.  (d) NetCache.

- 0.28->0.36: 768 register slots
- 0.36->0.46: 3072 register slots
Existing solutions and limitations

- Static binding of register memory
  - Merged in compilation time
  - P4Visor [CoNEXT’18]

- Ignore hardware constraints
  - DPDK, BMv2
  - Hyper4 [CoNEXT’16]

- No network-wide dynamic allocation
  - Allocation in a single switch
Realizing dynamic register memory allocation

- How to enable online allocation
- How to modify P4 programs
- How to maximize multiplexing benefits

Virtual register memory
P4VRM compiler
Dynamic allocation
NetVRM architecture
## Virtual register memory

- **Page tables**
- **Counter record**

### Diagram:

- **App 1**
- **App 2**
- **App 3**

#### Control Plane

- **Page Table**
- **Virtual Register Memory**
- **Counter Record**

#### Virtual Array

- **NetVRM**

#### Data Plane

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>app=1</td>
<td>offset=0, size=16k</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>16k</td>
<td>16k</td>
<td>16k</td>
<td>16k</td>
<td>16k</td>
</tr>
<tr>
<td>app=2</td>
<td>offset=16k, size=16k</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>32k</td>
<td>32k</td>
<td>32k</td>
<td>32k</td>
<td>32k</td>
</tr>
<tr>
<td>app=3</td>
<td>offset=32k, size=32k</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Counter Record

- `total_cnt`
- `hit_cnt`
Address translation

Translation formula

$$PA = \left(\frac{VA}{size}, VA \% size + offset\right)$$

physical array index

physical slot index

$$size = 2, offset = 1$$
$$VA = 5 \rightarrow PA = (2, 2)$$
NetVRM architecture

xx.p4vrm

P4VRM Compiler

Network

Dynamic Memory Allocation

NetVRM

zoom in

Control Plane

Data Plane

App 1

App 2

App 3

Virtual Register Memory

Run-time API

Network Data Plane

Network Control Plane
Problem formulation

Objective

- Maximize number of applications with satisfied utility target

Constraints

- Register memory constraints on each switch
Scope of dynamic resource allocation

Elastic applications
- Work with a variable amount of register memory
- Overcome insufficient register memory with a fallback mechanism

Inelastic applications
- Require a fixed amount of register memory
- Cannot work with less

benefit
support
Challenges for dynamic resource allocation

- Definition of application utility
- Unknown and dynamic utility functions
- Multiple paths of an application
- Memory hit ratio by default
Memory hit ratio by default

\[ \text{hit ratio} = \frac{m}{n} \]

- Application-agnostic
- Reflect application-level performance
- Computed online
Challenges for dynamic resource allocation

- Definition of application utility
- Unknown and dynamic utility functions
- Multiple paths of an application
- Memory hit ratio by default
- Online utility curve estimation
Online utility curve estimation

\[ c_f : \text{compensate for the diminishing return} \]
Dynamic resource allocation

- Definition of application utility
- Memory hit ratio by default
- Step 1: decompose over_mem and under_mem
- Step 2: over_mem -> under_mem

Multiple paths of an application

Network-wide register memory allocation
NetVRM architecture
P4VRM compiler

Step 1: developers extend .p4 to .p4vrm

- Mark the register arrays and related declarations as virtualized
P4VRM compiler

Step 1: developers extend .p4 to .p4vrm

Step 2: compiler outputs .p4 and .cpp

developers

heavy_hitter .p4
netcache .p4

... heavy_hitter .p4vrm
netcache .p4vrm

p4vm compiler

.p4 with VRM
.cpp for updates
Implementation

- 6.5 Tbps Intel Tofino switch
- Four emulated switches with four independent pipelines
- P4VRM compiler
  - built on Flex/Bison
Evaluation

- **Microbenchmark**
  - Control loop delay
  - Stability and fast convergence of NetVRM

- **Macrobenchmark**
  - Generality
  - Impact of allocation epochs
  - Impact of workload parameters
  - NetVRM in datacenter network
Evaluation

- Microbenchmark
  - Control loop delay
  - Stability and fast convergence of NetVRM

- Macrobenchmark
  - Generality
  - Impact of allocation epochs
  - Impact of workload parameters
  - NetVRM in datacenter network
Control loop delay

- One reallocation can be done in ~10 ms
- Reconfig dominates the control loop
Generality

- Each application has traffic from four switches independently
- Satisfaction ratio as the performance metric
- Alternatives: Equal-All, Equal-Active

- NetVRM outperforms alternatives on both the mean and the tail
- NetVRM is general to different network application types
Impact of allocation epochs

- A shorter allocation epoch leads to a better performance
Conclusion

- NetVRM supports dynamic register memory sharing between multiple concurrent applications on a programmable network
  - Virtual register memory: enable online register memory sharing
  - Dynamically allocate memory for better resource efficiency
  - P4VRM: easily equip the programs with virtual register memory
Thank you!

E-mail address: hzhu@jhu.edu