OvS Lookup Optimization
Using Two-Layer Table Lookup

Sameh Gobriel and Charlie Tai

Intel Labs
OvS De Facto Virtual Switch for NFV Environments

Monolithic Purpose-built Boxes

- Network appliances use purpose-built H/W & ASICs (e.g., TCAM) for flow classification
- Cost & power consumption are limiting factors to support large number of flows

Networking VMs on Standard Servers

- General purpose processors with Cache/memory hierarchy can support much larger flow tables.
- Multicores architecture provide a scalable competitive flow classification performance.
Open vSwitch Flow Lookup

1. Set of disjoint sub-table with no priority
2. Rule is only inserted into one sub-table (lookup terminates after first match)
3. Lookup is done by sequentially search each sub-table until a match is found

OvS Flow Classification is a bottleneck

Fig. Vtunes OVS flow lookup process (bypass EMC). Test case: 20 sub-tables, each has 100 rules.
Two Layer Table Lookup Abstraction for MFC

L Lookups → 1 lookup + 1st Level Indirection Overhead
Bloom Filter – Background

- With certain false positive rate, bloom filter is used to check if a variable (x,y,z) is a member. Member means the variable has been inserted already.

- We can use bloom filter to check if a flow is inside a sub-table or not, before searching the sub-table.
Bloom Filter – Lookup Scheme

- Before searching into sub-table, we use bloom filter to check if the masked key (sub-key) is a member of the sub-table or not.
Cycles Breakdown Assuming L-subtable traversal

Using Current Scheme

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
<th>Repetition</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O</td>
<td>210</td>
<td>1</td>
</tr>
<tr>
<td>Miniflow Extract</td>
<td>103</td>
<td>1</td>
</tr>
<tr>
<td>Hash For Submask</td>
<td>97 L</td>
<td></td>
</tr>
<tr>
<td>Subtable Sig. Cmp</td>
<td>53 L</td>
<td></td>
</tr>
<tr>
<td>Full Key Comparison</td>
<td>82</td>
<td>1</td>
</tr>
</tbody>
</table>

Hit Cycles = 395 + L * 150

Using Bloom Filter

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
<th>Repetition</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O</td>
<td>210</td>
<td>1</td>
</tr>
<tr>
<td>Miniflow Extract</td>
<td>103</td>
<td>1</td>
</tr>
<tr>
<td>Hash For Bloom Filter</td>
<td>88 L</td>
<td></td>
</tr>
<tr>
<td>Bloom Filter Lookup</td>
<td>30 L</td>
<td></td>
</tr>
<tr>
<td>Check Subtable Sig. Cmp</td>
<td>53</td>
<td>1</td>
</tr>
<tr>
<td>Full Key Comparison</td>
<td>82</td>
<td>1</td>
</tr>
</tbody>
</table>

Hit Cycles = 448 + L * 118
Vector Bloom Filter – Lookup Scheme

- **Vector Bloom Filter (or vBF)** hashes and stores unmasked full keys (like EMC).
- vBF Filter for each sub-table store encountered full keys corresponding to rules in sub-tables.
- A new flow always misses vBF (similar to EMC) but can hit a rule in the sub-table.
Vector Bloom Filter – Cost Analysis

Using Original Scheme

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
<th>Repetition</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O</td>
<td>210</td>
<td>1</td>
</tr>
<tr>
<td>Miniflow Extract</td>
<td>103</td>
<td>1</td>
</tr>
<tr>
<td>Hash For Submask</td>
<td>97</td>
<td>L</td>
</tr>
<tr>
<td>Subtable Sig. Cmp</td>
<td>53</td>
<td>L</td>
</tr>
<tr>
<td>Full Key Comparison</td>
<td>82</td>
<td>1</td>
</tr>
</tbody>
</table>

Hit Cycles = 395 + L * 150

Using vBF

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
<th>Repetition</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O</td>
<td>210</td>
<td>1</td>
</tr>
<tr>
<td>Miniflow Extract</td>
<td>103</td>
<td>1</td>
</tr>
<tr>
<td>Hash For XBloom (full Key)</td>
<td>159</td>
<td>1</td>
</tr>
<tr>
<td>Bloom Filter Lookup</td>
<td>30</td>
<td>L</td>
</tr>
<tr>
<td>Check Subtable Sig. Cmp</td>
<td>53</td>
<td>1</td>
</tr>
<tr>
<td>Full Key Comparison</td>
<td>82</td>
<td>1</td>
</tr>
</tbody>
</table>

Hit Cycles = 607 + L * 30

Packet header

Flow mask

11100000
11100000
11000000

101x xxx
001x xxx
101x xxx
00x xxx
10x xxx

Rules

1010 xxxx
0011 xxxx
1011 xxxx

101x xxxx
001x xxxx
101x xxxx
00x xxxx
10x xxxx

10xx xxxx
00xx xxxx
10xx xxxx
Results and performance figures are for an experimental prototype and is work in progress. The result reflect specific components on a particular test, in specific systems and should not be generalized for actual products. Differences in hardware, software, or configuration will affect actual performance.

Results are generated using a model based on processing cycles of Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz using OvS 2.6.0 with 20 sub-table and uniform random traffic.
Counting Bloom Filters to Handle Deletion

Start with an $m$ bit array, filled with 0s.

Hash each item $x_j$ in $k$ times. If $H_i(x_j) = a$, add 1 to $B[a]$.

To delete $x_j$, decrement the corresponding counters.

Can obtain a corresponding Bloom filter by reducing to 0/1.

4 bits/counter $\rightarrow$ Probability of Overflow = 6.78 E-17
Vector Bloom Filter – Results

- vBF provides significant lookup performance gain when compared with native OvS.
- Gain increases with increasing number of sub-tables.

Fig. 20 subtables, and each sub-table contains various numbers of rules. Note that we disabled EMC for showing the benefits.

Results and performance figures are for an experimental prototype and is work in progress. The results reflect specific components on a particular test, in specific systems and should not be generalized for actual products. Differences in hardware, software, or configuration will affect actual performance. Results are generated using uniform random traffic with 20 sub-tables running on Intel® Xeon(R) CPU E5-2699 v4 @ 2.20GHz and using OvS 2.6.0.
Conclusion

• Flow Lookup is a performance bottleneck for OvS, especially with increasing number of flows and sub-tables.

• Two layer table architecture optimizes flow lookup in OvS and avoids the sequential search of the sub-tables.

• Vector Bloom Filter (vBF) uses bloom filters as the first layer and can significantly improves lookup performance for OvS.

• Future Work:
  • Investigate other technologies to use as the first layer of indirection.
  • Realistic traffic pattern and workload
Questions?

sameh.gobriel@intel.com
charlie.tai@intel.com