High Performance Packet Forwarding with CuckooSwitch and Integration with Intel DPDK

Dong Zhou, Bin Fan, Hyeontaek Lim, David G. Andersen, Michael Kaminsky (Intel Labs)
Carnegie Mellon University

Ren Wang, Sameh Gobriel, Christian Maciocco, George Kennedy
Intel Labs

Venky Venkatesan, Bruce Richardson
CSIG
Agenda

- CuckooSwitch: a software switch that
  - uses DPDK as IO engine
  - can handle extremely large forwarding tables while offering line-rate throughput
- Integration with Intel DPDK framework
  - benefit: improve DPDK large table forwarding performance
- current status
Challenges

• Requirements for network switches
  more lookups per second into larger tables
Why Not Existing Solutions?

- CAM/TCAM
  - expensive
  - power hungry
  - very limited in size

- Hash tables in DRAM
  - slow
  - memory inefficient
    - hash collisions
    - pointer-swapping
Intel DPDK

- high-throughput packet I/O to userspace

- RX Queues
- TX Queues

- Ports
- Cores

- Optimized optimistic concurrent cuckoo hashing

- high performance switch FIB
Hash Table

- Desired properties
  - Fast
  - High occupancy
  - In-place update
Cuckoo Hashing

- Each key has two candidate buckets
  - Assigned by hash1(key) and hash2(key)
  - Stored in one of the candidate buckets
- Lookup: check two buckets in parallel
- Insert: perform key displacement recursively
Cuckoo Hashing [Pagh 01]

- Each key has two candidate buckets
  - Assigned by hash1(key) and hash2(key)
  - Stored in one of the candidate buckets
- Lookup: check two buckets in parallel

- Insert: perform key displacement recursively
- 95% occupancy when set-associativity is 4
Optimistic Concurrent Cuckoo Hashing\cite{NSDI 13}

- Higher concurrency
  - single-writer/multi-readers by optimistic concurrency control
Simplified Multi-Reader/Single-Writer

- Keep a version number for each bucket
- Lookup
  - $v = \text{bucket version before lookup}$
  - $v' = \text{bucket version after lookup}$
  - Compare $v$ with $v'$, retry if mismatch or $v$ is odd
- Insert
  - Increase versions of involved buckets for each displacement
System Optimizations

- We share the similar principles with DPDK, especially:
  - Batched hash table lookup with prefetching
Lookup throughput is very sensitive to memory access latency
Batched Lookup with Prefetching

hash1(keys[1])  hash1(keys[2])  hash1(keys[3])  hash1(keys[4])


CPU  DRAM
Batched Lookup with Prefetching

- Prefetch b1[1]
- Read check b1[1] b1[1]
- Prefetch b2[1]
- Prefetch returns
- Cache hit!
Batched Lookup with Prefetching

- Prefetch one bucket after hash computation
  - Interleave computation w/ memory accesses
  - Better use available execution units and CPU load buffers
- 1.5 cache-line retrievals on average
Performance Evaluation
Experiment Setup

10GbE Ports

Cores

PCIe Controller

DRAM

QPI

Intel Xeon E5-2680

PCIe Controller

DRAM
## Raw Packet I/O

<table>
<thead>
<tr>
<th>Packet Size (Bytes)</th>
<th>Throughput (Mpps)</th>
<th>Throughput (Gbps)</th>
<th>Bottleneck</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>92.22</td>
<td>61.97</td>
<td>PCIe B/W</td>
</tr>
</tbody>
</table>
# Raw Packet I/O

<table>
<thead>
<tr>
<th>Packet Size (Bytes)</th>
<th>Throughput (Mpps)</th>
<th>Throughput (Gbps)</th>
<th>Bottleneck</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>92.22</td>
<td>61.97</td>
<td>PCIe B/W</td>
</tr>
<tr>
<td>128</td>
<td>66.24</td>
<td>78.43</td>
<td>PCIe B/W</td>
</tr>
<tr>
<td>192</td>
<td>47.17</td>
<td>80</td>
<td>Network B/W</td>
</tr>
<tr>
<td>256</td>
<td>36.23</td>
<td>80</td>
<td>Network B/W</td>
</tr>
</tbody>
</table>
End-to-end Benchmark

Throughput (Mpps) vs. 
# of FIB entries

- optimized cuckoo
- dense_hash_map

500K, 1M, 2M, 4M, 8M, 16M, 32M, 64M, 125M, 250M, 500M, 1B

4.00GiB
8.00GiB
End-to-end Benchmark

Throughput (Mpps)

- optimized cuckoo
- dense_hash_map
- batched dense_hash_map

- thread-safe
- CRC32c
- not thread-safe
- h(x) = x

# of FIB entries

500K 1M 2M 4M 8M 16M 32M 64M 125M 250M 500M 1B

4.00GiB 8.00GiB 16.32GiB
End-to-end Benchmark

Throughput (Mpps)

# of FIB entries

2M

16M

concurrent cuckoo
hugepage
memorder
batching
prefetching

0 25 50 75 100

22
Integration with Intel DPDK
Intel DPDK

- User mode packet processing framework providing great IO performance.
- Packet flow classification library uses hashing to achieve line-rate packet switching.
- It is challenging to maintain line rate with current default hashing designs when number of flows grows large.
- CuckooSwitch hash table design naturally aligns with DPDK framework.
The benefit of DPDK with CuckooHashing

- CuckooHashing provides ~15X improvement on table efficiency.

- Reduces memory bandwidth due to higher cache utilization

- Maintains throughput with large table (64M entries) comparing to the current DPDK flow classification hashing.
Integration approach

- Include CuckooHashing as one option in DPDK flow classification library.
  - Working with CSIG closely
- Better address customers’ need
  - Especially Telco industry such as AT&T.
Current status

- Licensing
  - BSD licensing (thanks to ISTC 😊)
- Comply with DPDK framework
  - Unified APIs
  - Code optimization and performance evaluation
- Functionality extension
  - E.g., variable key-length hashing
- Architectural and system behavior characterization for understanding and optimization
  - Unit test and optimization for hashing functionalities
  - Architectural characteristics of hashing behavior
Conclusion

**DPDK**

- High performance IO framework
- Flow classification library for switching
- Build on top of DPDK framework

**CuckooSwitch**

- Cuckoo hashing handles large number of entries
- System optimizations that aligns with DPDK
- Integrating CuckooHashing into DPDK benefits Intel and industry
- Continue to collaborate towards future communication centric workload optimizations
backup
Intel DPDK framework
Where we can help with Cuckoo Hashing
Latency

The average latency is ~ 35 microsecond under maximum throughput