BMAcc: Accelerating P4-Based Data Plane with DPDK

PEILONG LI *, XIAOBAN WU *, YAN LUO *, LIANG-MIN WANG+, MARC PEPIN+, ATUL KWATRA+, AND JOHN MORGAN+

* UNIVERSITY OF MASSACHUSETTS LOWELL
+ INTEL CORPORATION

DPDK Summit - San Jose – 2017
Agenda

- Background of P4 and BMv2
- Problems and design motivations
- Overview of BMAcc & performance optimizations
- Performance evaluation
- Future directions
- Conclusion
P4 Language and P4 Behavior Model v2

- Programming Protocol-Independent Packet Processors (P4)
  - Simple semantics, customizable headers & dataplane functions
  - P4 program → P4 Frontend Compiler → Python IR → Backend Compiler → Target (software switch, NPU, etc.)

- P4 Behavior Model version 2 (BMv2)
  - BMv1 is deprecated: requires re-compilation for every P4 program.
  - BMv2 is a static executable: software switch consists of building blocks (parser, deparser, match-action tables, etc.)
  - Configured by JSON: P4 program → p4c-bm → JSON config → BMv2 (static)
Problems of BMv2

- A great software switch to verify the “behavior” of a P4 program
- Poor performance as a software switch
  - 99.993% packet drop rate with 64-byte packet on 10 Gbps link
  - Uses libpcap, Linux NIC driver, single-threaded, single RX queue (no RSS), unnecessary memory copy, etc.
Design an Accelerated Data Plane BMAcc for P4:
- Not a P4 compiler (e.g. PISCES\textsuperscript{[1]}, P4ELTE\textsuperscript{[2]}), a P4 target on multicore platforms.
- A substitute of BMv2 but with line rate performance.

Performance Acceleration:
- Leverages DPDK libraries and PMD driver for faster packet I/O.
- Applies multiple optimization techniques: reduced memory copy, multithreading, SSE instr.

Transparent to P4 Programs:
- Support all P4 programs and DPDK-compatible platforms.
- Not require P4 source code, only JSON config files.

\textsuperscript{[2]} Laki et al. 2016. High speed packet forwarding compiled from protocol independent data plane specifications. In SIGCOMM ’16.
The Design Overview

Three Optimizations

- Opt 1: ① PCAP → DPDK; ② Linux driver → PMD; ③ Single thread → RSS multi-queue
- Opt 2: ① rm redundant mem copy in Receive; ② rm MUTEX for each parsed header
- Opt 3: ① P4 LPM → DPDK LPM; ② SSE instructions used by DPDK.
Evaluation Setup

- 2 Intel Supermicro Servers [1] w/ 2 * 10 GbE NIC cards on each server.

- SM1:
  - TX: pktgen - 10 Gbps traffic with random dst IP
  - RX: count the received packets

- SM2:
  - BMv2 Simple Router target.
  - Lookup table, forward/drop.

[1] Intel Supermicro server 1U based on Intel® Xeon® processors D-1540 @ 2.0 GHz, Niantic 82599 10 GbE NIC.
Test 1: Hardware Performance Verification

NIC TX Capability:

- TX end always sends 64*1024*1024 packets with 256*1024 distinct flows

<table>
<thead>
<tr>
<th>Packet Size (byte)</th>
<th>Throughput</th>
<th>Framing rate</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>9.8 Gbps</td>
<td>9.9 Gbps</td>
<td>79 Sec</td>
</tr>
<tr>
<td>750</td>
<td>9.7 Gbps</td>
<td>9.9 Gbps</td>
<td>41 Sec</td>
</tr>
<tr>
<td>64</td>
<td>7.7 Gbps</td>
<td>9.9 Gbps</td>
<td>4.7 Sec</td>
</tr>
</tbody>
</table>

NIC RX Capability:

- RX end only receives packets and then drops.
- No transmission to the out-port.
Test 2: Performance on a Single Core with Different Optimizations

- Vanilla BMv2 only supports single thread.
- Performance comparison: PCAP (vanilla) → Opt1 → Opt 1,2 → Opt 1,2,3

1. Almost all DPDK versions outperform PCAP
2. More opts → higher performance
3. Opt1 single core version: DPDK has no evident performance gain
Test 3: Performance on 8 Cores with Different Optimizations

- PCAP is not on the chart
- Performance comparison: Opt 1 → Opt 1,2 → Opt 1,2,3

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>98.51%</td>
<td>99.25%</td>
<td>99.25%</td>
<td>65.72%</td>
<td>94.19%</td>
<td>99.25%</td>
</tr>
<tr>
<td>750</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>7.15%</td>
<td>10.73%</td>
<td>10.73%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. 3x, 5.5x, 23x increase over single core vanilla BMv2 for large, mid, and small packet sizes
2. Reach line rate for large and mid sized packets with 3 opts
3. How about 64-byte packet?
Test 4: Find the Performance Killer for Small Packets

- Five major stages in P4 Processing:
  - RX → Parser → LPM → Deparser → TX
- Gradually add stages to this pipeline to find the biggest performance drop
- In experiment: 4 Cores, 64-byte packet

1. Perf impact breakdown:
   - TX: 20%
   - Parser: 58%
   - Deparser: 5%
   - LPM: 9%
2. TX+RX → Similar to l3fwd (80% PRR as reported)
3. Parser – creates NEW objects for each packet → time consuming
Take the Opt 1,2,3 case (the most optimized)

1. Large packet reaches line rate w/ 4 cores; mid packet w/ 8 cores
2. Performance is almost proportional to # of cores
3. Not shown here, but the results are consistent with Opt 1 and Opt 1-2.
Test 6: The Performance of LPM Processing

- P4 LPM: leverages Judy for creating and accessing dynamic arrays
- DPDK LPM: SSE instructions and cache friendly data structures

1. DPDK-LPM is slightly better for all cases
2. DPDK-LPM performance benefit is more evident when ruleset is smaller and processing cores are fewer because of the overhead of Judy library.
The DPDK-accelerated BMv2 reaches 10 Gbps line rate for mid & large-sized packets, and yields 23x performance boost on the small packets.

To address the Parser impact on 64-byte packet, we need to pre-allocate memory spaces for Packet instances.

We proposed multiple practical optimizations on the BMv2 which are instrumental to all P4-based data plane designs on multicore platforms.

We conducted in-depth performance study on the proposed BMAcc system from architecture and software perspectives.
Questions?

Peilong Li, Ph.D.
Research Assistant Professor
https://peilong.github.io

UMass Lowell ACANETS Lab
http://acanets.uml.edu