NDN-DPDK: NDN Forwarding at 100 Gbps on Commodity Hardware

Junxiao Shi, Davide Pesavento, Lotfi Benmohamed
Advanced Network Technologies Division
National Institute of Standards and Technology
Introduction

• NDN needs a high-speed forwarder:
  • Use case: data intensive science, live video streaming, ...
• Goal: line speed on commodity hardware.

How to get there?
➢ Adopt better algorithms and data structures.
➢ Reduce overhead in library and kernel.
Data Plane Development Kit (DPDK)

• DPDK: libraries to accelerate packet processing workloads.

Main DPDK features:

➢ Multi-threading: use all available CPU cores.
➢ Ring buffer queue: transfer packets between threads.
➢ Hugepage-backed memory pools: no malloc() in data path.
➢ User-space NIC drivers: bypass the kernel.
Our Contributions

• **NDN-DPDK:**
  ✓ Complete implementation.
  ✓ Running on real hardware.
  ✓ Support full NDN protocol and name matching semantics.

• **Prior works:**
  ❑ Focus on a subset of data plane: Mansilha et al (ICN'15), ...
  ❑ Rely on simulations: Song et al (ICN'15), ...
  ❑ Lack support for Interest-Data prefix match: So et al (ANCS'13), Caesar (ANCS'14), Augustus (ICN'16), ...
Forwarder Architecture

input0

input1

fwd0

PIT+CS

FIB

fwd1

PIT+CS

FIB

output0

output1

Name Dispatch Table

FIB mgmt

input stage

forwarding stage

output stage
FIB Design

- 2-stage Longest Prefix Match algorithm.
  - So et al, Named data networking on a router: Fast and DoS-resistant forwarding with hash tables (ANCS'13).
FIB Replication on NUMA Sockets

• NUMA: Non-Uniform Memory Access.
  • Hardware in a multi-CPU server is organized in NUMA sockets.
  • Nonlocal memory access incurs higher latency.

• Each NUMA socket has a copy of FIB.
  • Forwarding threads can avoid nonlocal memory access during FIB lookups.
PIT Sharding

• Each forwarding thread has a private PIT.
  • Non-thread-safe. No RCU.

Requirements on packet dispatching:

1) Same Interest name => same forwarding thread.
   • Required by Interest aggregation and loop prevention.

2) Common Interest prefix => same forwarding thread.
   • Make forwarding strategy effective.

3) Data/Nack => forwarding thread that processed the Interest.
   • So that they can go back to the downstream.
Dispatch Interest by Name

• Name Dispatch Table (NDT)
  • Map: hash of name prefix => forwarding thread ID
  • Thread safe: NDT is an array of atomic_int.
  • Many name prefixes share the same entry.

• In input threads:
  1. Compute hash of first two name components (configurable)
  2. Choose NDT entry: hash % NDT.size()
  3. Dispatch to forwarding thread

Diagram:
- Interest
- NDT Array
- fwd0 PIT+CS
- fwd1 PIT+CS
- Dispatch to forwarding thread
PIT Token

- Data packet: name dispatching works most of the time, except:
  - Interest /A CanBePrefix=1 goes to NDT[SipHash(/A)].
  - Data /A/B/1 goes to NDT[SipHash(/A/B)].

- Solution: use PIT token to associate Interest and Data.

- PIT token is an opaque token carried in a hop-by-hop field.
  - Every outgoing Interest carries a PIT token.
  - Data/Nack must carry the same PIT token.
Dispatch Data/Nack by PIT Token

- NDN-DPDK's PIT token contains:
  a) Forwarding thread ID (8 bits), to dispatch Data/Nack correctly.
  b) PIT entry index (48 bits), to accelerate PIT lookups.
Prefix Match in CS

• In-Network Name Discovery:
  • Interests should be able to use incomplete names to retrieve Data packets.
• CS is a hash table, which only supports exact match.
• Solution: indirect entries.

Prefix-name Interest /A/B can be satisfied by the indirect entry.

Consumer normally uses a consistent name for name discovery. If a different name is used, the Interest must be satisfied by the producer, and then it gets another indirect entry.

Exact-name Interest /A/B/C/1 can be satisfied by the direct entry.

/A/B/C/1 direct entry with Data packet

/A/B/C indirect entry
PIT-CS Composite Table (PCCT)

- PIT entry:
  - copy of Interest
  - downstream faces
  - upstream faces
  - strategy scratch area

- direct CS entry:
  - Data packet
  - indirect entries list
  - ARC linked list nodes

- indirect CS entry:
  - direct entry pointer
  - LRU linked list node

- Name hash table:
  - name
  - chosen FH
  - 48-bit token

- Token hash table:
  - name
  - chosen FH
Benchmarks

Spoiler alert: we made it to 100 Gbps
Benchmark Topology

Two physical machines:
• Forwarder.
• Traffic generators: (logically independent)
  • Fetch Data from each other.
  • CUBIC-like congestion control.

• CPU: dual Intel Xeon Gold 6240.
  • 18 cores at 2.60 GHz, Hyper Threading disabled.
• Memory: 256 GB, 2933 MHz, four channels.
  • 64x 1GB hugepages per NUMA socket.
• NIC: Mellanox ConnectX-5 100 Gbps.
We Made It to 100 Gbps

Measured from consumers. Data packets only. Not counting retransmissions.

Data payload only.

8 forwarding threads.
Input Thread Bottleneck

• Expectation:
  ▲ # forwarding threads
  ▲ Data forwarding rate (pps)

• Reality:
  • Data forwarding rate plateaus at 8 forwarding threads.

• Bottleneck: input thread.
  • Current architecture only allows one input thread per face.

<table>
<thead>
<tr>
<th>Name components</th>
<th>Forwarding threads</th>
<th>Data forwarding rate [Mpps]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>1.84 (peak)</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

linear growth    no improvement
Effect of Nonlocal Memory Access

- same NUMA
- cross NUMA: higher memory access latency

up to 20% slower

![Data forwarding rate vs. Forwarding threads](chart.png)
Performance with Large FIB

<table>
<thead>
<tr>
<th>FIB entries</th>
<th>Forwarding rate (kpps)</th>
<th>Interest latency (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>stdev</td>
</tr>
<tr>
<td>$10^4$</td>
<td>1840</td>
<td>5.59</td>
</tr>
<tr>
<td>$10^5$</td>
<td>1835</td>
<td>4.92</td>
</tr>
<tr>
<td>$10^6$</td>
<td>1839</td>
<td>4.42</td>
</tr>
</tbody>
</table>
Forwarding Rate with Large CS

![Graph showing data forwarding rate vs. content store hit ratio with different capacities and match types.]

- **cap=2^{17} match=exact**
- **cap=2^{20} match=exact**
- **cap=2^{17} match=prefix**
- **cap=2^{20} match=prefix**

Legend:
- **producer**
- **forwarder**
- **consumer**
- **consumer**
Latency with Large CS

**Exact Match**

- `cap=2^{17} - Data`
- `cap=2^{17} - CS miss`
- `cap=2^{17} - CS hit`
- `cap=2^{20} - Data`
- `cap=2^{20} - CS miss`
- `cap=2^{20} - CS hit`

**Prefix Match**

- `cap=2^{17} - Data`
- `cap=2^{17} - CS miss`
- `cap=2^{17} - CS hit`
- `cap=2^{20} - Data`
- `cap=2^{20} - CS miss`
- `cap=2^{20} - CS hit`

### 90th percentile

<table>
<thead>
<tr>
<th>Interest</th>
<th>4875 μs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>456 μs</td>
</tr>
</tbody>
</table>
Future Work

• Remove the input thread bottleneck:
  • Design changes to allow multiple input threads per face.
  • Dispatch Data/Nack via Receive Size Scaling (RSS).
  • Dispatch Interest (NDT) using eBPF or FPGA hardware.

• Expand Content Store to NVMe disk storage.

• Load balancing by adjusting NDT entries.

• Performance profiling and improvement toward 200 Gbps.
NDN-DPDK Codebase

- [https://github.com/usnistgov/ndn-dpdk](https://github.com/usnistgov/ndn-dpdk)
  - Forwarder
  - Traffic Generator
  - GraphQL-based management tools
  - NDNgo library for application development

- Dedicated to public domain
Thank You

Junxiao Shi, Davide Pesavento, Lotfi Benmohamed
NDN-DPDK: NDN Forwarding at 100 Gbps on Commodity Hardware
7th ACM Conference on Information-Centric Networking (ICN 2020)