High Performance Packet Processing with FlexNIC

Antoine Kaufmann, Naveen Kr. Sharma
Thomas Anderson, Arvind Krishnamurthy

Simon Peter

University of Washington  The University of Texas at Austin
Networks: Fast and Growing Faster

Ethernet Bandwidth [bits/s] vs Year of Standard Release

- 100 MbE
- 1 GbE
- 10 GbE
- 100 GbE
- 40 GbE
- 400 GbE

Year of Standard Release:
- 1990
- 1995
- 2000
- 2005
- 2010
- 2015
- 2020
Networks: Fast and Growing Faster

- 100 MbE
- 1 GbE
- 10 GbE
- 100 GbE
- 40 GbE
- 400 GbE

Year of Standard Release:

- 1990
- 1995
- 2000
- 2005
- 2010
- 2015
- 2020

Ethernet Bandwidth [bits/s]:

- 100 M
- 1 G
- 10 G
- 100 G
- 1 T

5ns inter-arrival time for 64B packets at 100Gbps
... but Packet Processing is Slow

• Many cloud apps dominated by packet processing
  • Key-value store, real-time analytics, intrusion detection

•Recv+send network stack processing overheads
  • Linux: 3.4µs
  • Kernel bypass: 1.4µs
  • Can parallelize, but still too slow

•RDMA
  • Difficult to traverse/modify complex data structures
  • Not well matched to client-server cloud apps
NIC & SW are not well Integrated

- Wasted CPU cycles
  - Packet parsing and validation repeated in software
  - Packet formatted for network, not software access

- Poor cache locality, extra synchronization
  - NIC steers packets to cores by connection
  - Application locality may not match connection
FlexNIC:
A Model for Integrated NIC/SW Processing

- Must be implementable at line rate with low cost
- Match+action pipeline:
Match+Action Programs: Actions

**Match:**
IF udp.port == kvs

**Action:**
core = HASH(kvs.key) % 2
DMA hash, kvs TO Cores[core]

**Supports:**
- Steer packet
- Calculate hash/Xsum
- Initiate DMA operations
- Trigger reply packet
- Modify packets

**Does not support:**
- Loops
- Complex calculations
- Keeping large state
FlexNIC: M+A for NICs

- Efficient application level processing in the NIC
  - Improve locality by steering to cores based on app criteria
  - Transform packets for efficient processing in SW
  - DMA directly into and out of application data structures
  - Send acknowledgements on NIC
Outline

• Motivation

• FlexNIC Programming model

• Key-Value Store
  • Optimizing Reads: Key-based Steering
  • Optimizing Writes: Custom DMA Interface

• Real-time Analytics

• Intrusion Detection System

• Performance Evaluation for Key-value Store

• Limitations and Future Work
Example: Key-Value Store

Client 1
K = 3, 4

Client 2
K = 1, 4, 7

Client 3
K = 1, 7, 8

NIC

Hash Table

Core 1

Core 2
Example: Key-Value Store

Receive-side scaling:
core = hash(connection) % N

Client 1
K = 3, 4

Client 2
K = 1, 4, 7

Client 3
K = 1, 7, 8

Hash Table
Example: Key-Value Store

**Receive-side scaling:**
core = hash(connection) % N

Client 1
K = 3, 4

Client 2
K = 1, 4, 7

Client 3
K = 1, 7, 8

Hash Table

Core 1

Core 2

4

7
Example: Key-Value Store

Receive-side scaling:
core = hash(connection) % N

- Lock contention
- Poor cache utilization
Optimizing Reads: Key-based Steering

Client 1
K = 4, 3

Client 2
K = 1, 4, 7

Client 3
K = 1, 7, 8

NIC

Hash Table

1 2 3 4 5 6 7 8

Core 1

Core 2
Optimizing Reads: Key-based Steering

**Match:**
IF udp.port == kvs

**Action:**
core = HASH(kvs.key) % 2
DMA hash, kvs TO Cores[core]

**Client 1**
K = 4, 3

**Client 2**
K = 1, 4, 7

**Client 3**
K = 1, 7, 8

**NIC**

**Hash Table**
1
2
3
4
5
6
7
8
Optimizing Reads: Key-based Steering

**Match:**
IF udp.port == kvs

**Action:**
core = HASH(kvs.key) % 2
DMA hash, kvs TO Cores[core]

<table>
<thead>
<tr>
<th>Client 1</th>
<th>K = 4, 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client 2</td>
<td>K = 1, 4, 7</td>
</tr>
<tr>
<td>Client 3</td>
<td>K = 1, 7, 8</td>
</tr>
</tbody>
</table>

NIC

Core 1

Core 2

Hash Table

1
2
3
4
5
6
7
8
Optimizing Reads: Key-based Steering

Match:
IF udp.port == kvs

Action:
core = HASH(kvs.key) % 2

• No locks needed
• Better cache utilization

Client 1
K = 4, 3

Client 2
K = 1, 4, 7

Client 3
K = 1, 7, 8

NIC

Hash Table
1 2 3 4 5 6 7 8
Optimizing Writes: Custom DMA

- DMA to application-level data structures
- Requires packet validation and transformation
Optimizing Writes: Custom DMA

- DMA to application-level data structures
- Requires packet validation and transformation

Event Queue

G

GET, Client ID, Hash, Key

Item Log

Item 1
Optimizing Writes: Custom DMA

- DMA to application-level data structures
- Requires packet validation and transformation

Diagram:
- Event Queue
  - G
  - S
  - SET, Client ID, Item Pointer
- Item Log
  - Item 1
  - Item 2
Real-time Analytics System

- Offload (de)multiplexing and ACK generation to FlexNIC
Real-time Analytics System

- Offload (de)multiplexing and ACK generation to FlexNIC
Snort Intrusion Detection

- Snort: Sniffs packets and analyzes them
- Parallelized by running multiple instances
- Status quo: Receive-side scaling for spreading to cores

- FlexNIC:
  - Analyze rules loaded into Snort
  - Partition rules to cores
  - Fine-grained steering to cores
Evaluation of the Model

• Measure impact on application performance
  • Without waiting for hardware implementation

• Re-use existing NIC functionality
  • Hash on certain fields

• Software emulation of M+A pipeline

Key-value store:

• Workload: 100k 32B keys, 64B values, 90% GET
• 6 Core Sandy Bridge Xeon 2.2GHz, 2x 10G links
Key-based steering

- Better scalability
  - PCIe is bottleneck for 4+ cores
- 30-45% higher throughput
- Processing time reduced from 510ns to 310ns
Key-based steering

- Better scalability
- PCIe is bottleneck for 4+ cores
- 30-45% higher throughput
- Processing time reduced from 510ns to 200ns

Steering and custom DMA reduces time from 510ns to 200ns
Ongoing Work/Limitations

• End to end validation
  • Netronome, FPGA NIC: validate performance with PCIe

• Hardware validation
  • Can match+action be implemented cheaply at line rate?

• Programming model for mixed NIC/SW processing
  • Draw inspiration from P4 and click

• Secure isolation between multiple applications
  • Currently the kernel mediates M+A installation
Summary

• Networks are becoming faster
  • Server applications need to keep up
  • Fast I/O requires fine-grained app-level I/O control

• FlexNIC model can eliminate inefficiencies
  • Application control over where packets are processed
  • Efficient steering/validation/transformation

• Case study: Key-value store
  • 30-45% throughput speed-up
  • 60% processing time reduction