Embedded Network Architecture Optimization Based on DPDK

Lin Hao
T1 Networks
Our History — What is an embedded network device
Challenge to us — Requirements for device today
Our solution — T1 unique embedded network architecture (T1-System)
  - Model of “embedded network architecture”
  - History of T1-system
  - Business layer of T1-system
  - An optimization case — dual-socket system
  - T1-system as a NFV
Our History

T1 Networks —
“Professional application delivery & High-performance fusion of network security products”

Harbor Networks Corp.
- **Product**: Router
  - **HW**: Freescale + Intel NP
  - **SW**: vxworks + uCode

Venustech Corp.
- **Product**: UTM
  - **HW**: Cavium OCTEON
  - **SW**: cvm execuitiveSDK

T1 networks Corp.
- **Product**: ADC
  - **HW**: X86
  - **SW**: Linux+Netmap
- **Product**: NG FW
  - **HW**: X86
  - **SW**: Linux+DPDK
Challenge to our system

**Situation**

1. **Falling cost on network bandwidth**
   - 10Gbps
   - 100Gbps
   - 40Gbps
   - 10/100/1000 Mbps

2. **Hardware is varied and iteration fast**
   - Xeon
   - Atom
   - I350
   - XL710
   - RRC
   - X552

3. **Features expansion**
   - VPN
   - Anti-virus
   - QoS
   - IPS
   - Compress

**require for our system**

- **Performance**!
- **Compatibility**!
- **Scalability**!
Model of ENA

Model of "Embed Network Architecture"

- **Network traffic**
- **System components**
- Be responsible for "Scalability"
- Be responsible for "Performance"
- Be responsible for "Compatibility"

**Hardware adaptation layer**
- Mbuf
- Ring
- Mem
- Timer
- Packets IO

**Traffic distribute layer**
- HW Drivers

**Business Layer**

**HW**
- I350
- 82599
- XL710

**SW**
History of “T1-system”

- 1st Generation —— “kernel driver based” system
- 2nd Generation —— Muti-Core MIPS64
- 3rd Generation —— “Dispatcher-application” system
- 4th Generation —— “Balanced-dispatcher” DPDK-equipped system
- 5th Generation —— “DPDK+FPGA” system

Why we need DPDK? How to use DPDK?
1st Gen—Kernel driver based

**Advantage:**
Easy to get......

**Problem:**
① Bottleneck of Linux IRQ
② Difficult to develop and optimize
③ Inefficient system call
2nd Gen—Muti-Core MIPS64

**Advantage:**
Excellent throughput performance

**Problem:**
1. Performance decline on complex feature
2. Hard to develop
3rd Gen—Dispatcher-Application

1. RSS-bound packets handle
2. 5-tuple hash dispatcher
3. control-plane Vs data-plane
3rd Gen—Dispatcher-Application

**Advantage**: Reduced Muti-core competition
3rd Gen—Dispatcher-Application

**Problem:** Bottleneck in different situations

- **Heavy traffic, low complexity:** Bottleneck
- **Light traffic, high complexity:** Inefficient
4th Gen—DPDK-equipped system

Two improvement:

1. DPDK-equipped.
2. “balance-dispatcher” system.
5th Gen—Maybe in the future

Core group: ctrl_plane

Single core with DPDK

APP-D

DPDK

Intel NIC

RSS

Intel FPGA

Dispatcher

Release CPU cost from dispatcher.

Avoid packets deliver between cores.

More efficient on cache scheduling.
Why dispatcher in software

Can not use RSS hash, why?

Precondition:
1. HASH value of both sides must be consistent
2. port_D2 can be decided

Calculate process:

\[
\text{HASH\_VALUE} = \text{hash}(\text{IP}\_C, \text{port}\_C, \text{IP}\_D1, \text{port}\_D1)
\]

\[
\text{port}\_D2 = \text{hash\_inverse}(\text{HASH\_VALUE}, \text{IP}\_D2, \text{IP}\_S, \text{port}\_S)
\]
History of “T1-system”

- 1st Generation — “kernel driver based” system
- 2nd Generation — Muti-Core MIPS64
- 3rd Generation — “Dispatcher-application” system
- 4th Generation — “Balanced-dispatcher” DPDK-equipped system
- 5th Generation — “DPDK+FPGA” system

**Why we need DPDK? How to use DPDK?**
Why DPDK??

DPDK vs netMap

1. **Performance**: E5-2670V3 24 cores/1000 policies/64-bytes throughput

<table>
<thead>
<tr>
<th></th>
<th>Throughput 64bytes</th>
<th>Latency Average (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netmap</td>
<td>27 Gbps</td>
<td>43700</td>
</tr>
<tr>
<td>DPDK</td>
<td>102.4 Gbps</td>
<td>20601</td>
</tr>
</tbody>
</table>
Why DPDK??

DPDK vs netMap

2. **Performance**: CPU cost analysis by oprofiler

<table>
<thead>
<tr>
<th>Function</th>
<th>DPDK lib</th>
<th>Netmap lib</th>
</tr>
</thead>
<tbody>
<tr>
<td>51544 18.2802 ipv4_rcv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38557 13.6743 se_resolve_normal_ct.part.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28482 10.1012 tb_skb_rcv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27258 9.6671 se_ip_contrack_in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25112 8.9060 tb_nf_hook_slow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11180 3.9650 recv_raw_pkts_vec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10610 3.7629 tb_skb_send</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9838 3.4891 xgbe_xmit_pkts_vec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9603 3.4057 __se_ip_ct_refresh_acct</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8172 2.8982 packet_intercept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7916 2.8074 tb_clear_skb_header</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7579 2.6879 jhash_3words</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4600 1.6314 tb_stat_flow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4073 1.4445 tb_skb_capture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3683 1.3062 tb_packet_handle_loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2918 1.0349 tb_rte_memcpy_func.constprop.23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2537 0.8998 tb_flow_stat_policy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33951 12.7653 se_resolve_normal_ct.part.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32165 12.0937 packet_intercept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26541 9.9792 se_ip_contrack_in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24187 9.0941 nm_send</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22004 8.2733 tb_nf_hook_slow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19018 7.1506 nm_recv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11845 4.4556 app_interface_flow_stat_entry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9797 3.6836 __se_ip_ct_refresh_acct</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8236 3.0967 tb_clear_skb_header</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7830 2.9440 jhash_3words</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6416 2.4124 nm_send_skb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6007 2.2586 ipv4_rcv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5821 2.1886 tb_skb_rcv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5576 2.0965 tb_packet_handle_loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5227 1.9653 tb_skb_xmit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4764 1.7912 tb_stat_flow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3728 1.4017 tb_skb_capture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2864 1.0768 tb_rte_memcpy_func.constprop.23</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Why DPDK??

**DPDK vs netMap**

3. Code maintenance costs

User application with DPDK

Intel DPDK PMD

User application with Netmap

Netmap IO Libs

User

Kernel

Linux UIO driver

Kemel

E1000E-netmap

IGB-netmap

IxBG-netmap

I40e-netmap

IG BVF-netmap

Other drivers

: Code block we should take care of
Application with DPDK

- **SW routine**
- **Execute flow**
- **DPDK functions**
- **Pkt flow**
- **Pkt queue**

- **Get Pkts from rNIC**
- **dispatch Pkts**
- **Get Pkts from vNIC**
- **Pkts Handle**
- **Flush Pkts to vNIC**
- **Flush Pkts to rNIC**

- **Dispatcher**
- **APP**
- **DPDK RX**
- **DPDK TX**
- **DPDK MBUF**
Agenda

- **Our History** — What is an embedded network device
- **Challenge to us** — Requirements for device today
- **Our solution** — T1 unique embedded network architecture (T1-System)
  - Model of “embedded network architecture”
  - History of T1-system
  - **Business layer of T1-system**
  - An optimization case —— dual-sockets system
  - T1-system as a NFV
Business layer of T1-System: Multi-path traffic handle system

Aim of Multi-path system: Reduce CPU cost on traffic processing.
Agenda

- **Our History** — What is an embedded network device
- **Challenge to us** — Requirements for device today
- **Our solution** — T1 unique embedded network architecture (T1-System)
  - Model of “embedded network architecture”
  - History of T1-system
  - Business layer of T1-system
  - **An optimization case —— dual-sockets system**
  - T1-system as a NFV
Optimization on Dual-sockets platform

Basic environment:
1. Separated buffers and queues initialization on each Numa node
2. Ethernet ports bind with a single-node.
3. In case of simple handle of packets, such as IP forwarding.

Case 1: Packets cross-QPI
Optimization on Dual-sockets platform

Basic environment:
1. Separated buffers and queues initialization on each Numa node
2. Ethernet ports bind with a single-node.
3. In case of complex handle of packets, such as traffic audit.
Agenda

- **Our History** — What is an embedded network device
- **Challenge to us** — Requirements for device today
- **Our solution** — T1 unique embedded network architecture (T1-System)
  - Model of “embedded network architecture”
  - History of T1-system
  - Business layer of T1-system
  - An optimization case —— dual-sockets system
- **T1-system as a NFV**
  - NFV resource pool
  - Fusion gateway
  - New solution: OVS with DPDK
NFV Case 1: NFV Resource pool

**NFV Resource pool:**

1. Multiple NFVs for each guest
2. Traffic between NFVs in the same guest is forwarding by HW switch fabric
3. Traffic is isolated by vlan tag between guests

**Scene:**
Multi-tenant in data-center/ same flow: define template for each tenant/Elastic expansion
NFV Case 2: Fusion gateway

Fusion gateway:

1. Pass through mode for IO Virtualization
2. Flexible flow-define rules:

   FW  ---|  ADC  ---|  WAF
     |      |      |
     |      |      | P2
     |      |      |
     |      |      |
   P1

Scene:

Gateway position/Face to network/High performance/Feature fusion

User defined flow rules, for example:
- Rule 1: Trunk1 <-> Trunk2, tag value=100
- Rule 2: Trunk2 <-> Trunk3, tag value=101
- Rule 3: p1 <-> trunk1, tag value=200
- Rule 4: p2 <-> trunk3, tag value=201
## About NFV-Comparison

Comparison of two scenarios

<table>
<thead>
<tr>
<th>IO Virtualization</th>
<th>Face to</th>
<th>performance requirement</th>
<th>number of VMs</th>
<th>Configuration focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFV resource pool</td>
<td>VF(SR-IOV)</td>
<td>Guest</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Fusion gateway</td>
<td>Passthrough</td>
<td>Network</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>

**Limitation**: Rely on Hardware fabric
New solution — OVS with DPDK

OVS with DPDK is a low cost, more flexible alternative.

Intel NICs with SRIOV
Thank you!