CONTAINER NETWORKING SOLUTIONS

Anjali Singhai Jain, Nupur Jain, Intel
Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.
Agenda

Introduction
- Container Network Functions & Interfaces
- Limitations
- Container Interface Classifier
- Community Solutions
- Our Proposed Solution
- Consolidation

Functionality Offload
- P4 Sample - Kubeproxy
- P4 Connection tracking
- P4 L4-L7 Classifier
Container Networking

1. Edge Router & LB

2. Cluster Router

3. Node: LB, DNAT, CT

4. L4-L7 classifier

5. Container Interface
Current State of Deployment

Need for Scale
CSP Container Scale Needed ~10K

Device performance
Throughput, Programmability, Resource Scaling has increased significantly

Effective utilization
of the system resources (cores, memory, network) to improve performance and packing of containers

Ideal Container Network Solution
5. Existing interfaces for containers

**Shared (Pod Interfaces)**

- Stacked netdevs on a PCIe PF netdev
- Assigned to container namespaces
- Examples: MACVLAN, IPVLAN, bridge

**Dedicated (Pod Interfaces)**

- SR-IOV VFs
- Too heavy
  - Separate PCIE config space
  - HW based packet replication for broadcast, multicast – higher PCI BW utilization
Why Hardware Accelerate?

End to end Maximize throughput

- Avoid the SW long path which limits how much a Server Pod can handle.

End to end Native Scale out

- By Reducing Latency and Jitter introduced by kernel to user context switch in present AF_XDP memory model.
- Dedicated resources takes away the need for OS to schedule on a shared resource. OS overhead for managing resources is gone.
  - CPU scheduling, memory management etc.

Security & Isolation

- Queue level isolation.
Assignable Container interfaces using X-IOV (S-IOV) & User Interrupts

Hardware Assisted Virtualization

Highly scalable and high-performance sharing of I/O devices across isolated domains

Assignable device Interfaces => User Container Interfaces

Platform Scalability using PASID

Support Virtual Device Composition

*RID and *PASID identifies the address space associated with the request

*ADI Memory Mapped regions

*RID = Requester ID

*PASID = Process Address Space Identifier

*ADI Assignable Device interfaces

SIOV Spec : Reference Number: 337679-001, Revision: 1.0
4. Container Interface Classifier: Solution

• L4-L7 Classifier and forwarder in the HW
• This extends the HW Offloaded vSwitch Classification End Point to the Container Interface.

Option 1: HW Offload the classifier & forwarder
  - AF_XDP raw_socket bound to a HW vPort/QP through Side band filters.
  - Provide inline filters to be added in HW as part of TX packet.
    - ATR style in ADQ

Option 2: HW Offload the Classifier
  - Provide a meta data classification hint to kernel/user with a packet.
    - Flow mark or a 32bit hash value based on L4-L7 fields.
3. Node: Load Balancer, DNAT, CT

Existing Solutions

Kube-proxy

- Kernel Netfilters - Not performant
  - Iptables $O(n)$ chains proportional to size of cluster, in-place rule modifications not possible.
  - IPVS $O(1)$ - hash ipset but do not work well with other services requiring iptables for filtering

- Kernel with eBF/XDP - Accelerated

Connection Tracking

- Robust to syn floods but limited by max size
- No flexibility, fixed hash algorithm and field selection for hash

Kernel Overall not very flexible, latencies due to irq processing, context switching, slow API configuration interfaces
Community’s Approach - eBPF/XDP

**Benefits** - Performant than the kernel

Designed as an alternative to DPDK.

Flexibility, code injected into the kernel

Ability to reload programs on-the-fly

**Network Functions** - Network Policy, Encryption, Load Balancer, Firewall, Monitoring etc

**Plugins** - Cilium, Calico, Katran (facebook), etc

**Limitations** -

- Not HW offloadable to ASICs.
  - Note: Netronome uses NPU general Purpose cores
- More cores required to scale connections
- Kernel/user space context switching
- General purpose CPUs and Memory architecture is not ideal for Deep table Lookup

**Solution** -

- Purpose built ASICs, purpose -built cores and dedicated Context aware caches may be the way to go
eBPF Implementation – Cilium

- Cilium does connect-time load balancing by hooking into the kernel XDP/TC hook on the receive.
- When a program tries to connect to a Kubernetes service, Cilium intercepts the connection attempt, load balances with DNAT's to directly connect to the backend pod's IP instead.

Throughput in queries/s – test run with fortio and ngnix. 2 clients, 2 NodePort IPs, 2 backend Pods per NodePort

<table>
<thead>
<tr>
<th></th>
<th>No XDP, 2 CLV interfaces</th>
<th>XDP, 1 CLV interface</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fortio + Nginx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Client 1</td>
<td>Client 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100B</td>
<td>47578.6</td>
<td>50411.5</td>
</tr>
<tr>
<td>1500B</td>
<td>46174.4</td>
<td>49990.4</td>
</tr>
</tbody>
</table>

No-XDP and XDP performed similar with 100,000+ connections. CPU consumption in both reaches to >8 cores with more sessions.

We suspect, XDP path based on this experiment, does not take advantage of hardware XDP_REDIRECT queue designed to send packets to another interface, hence no significant performance gains.
Our Approach – The Whole Datapath

*Programmable MAT tables. Contract between control plane and data plane for runtime control

*Device capability defined by architecture. Eg. wildcard match -TCAM, Exact match-SRAM. Compiler responsible for mapping.

*Parallel lookups, conditional actions & atomicity

*Features are defined in the software. Faster introduction, verification, test and deployment.

*Programmer defined 1) parse graph; headers and orders. 2) Packet modifications

*Counters, meters, stateful registers, hash functions, ALU, TTLs, PRE
Node LB Data Plane P4

<table>
<thead>
<tr>
<th>Table Entry</th>
<th>Key/ClusterIP, port</th>
<th>Member group</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>10.3.241.152</td>
<td>g1</td>
</tr>
<tr>
<td>r2</td>
<td>10.3.241.159</td>
<td>g2</td>
</tr>
<tr>
<td>r3</td>
<td>10.3.241.170</td>
<td>g3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group/Port</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>m1, m2, m3</td>
</tr>
<tr>
<td>m2</td>
<td>m2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memberport</th>
<th>Action Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>10.0.1.2, 8080, 02, 42, 88, 7e, 7f, 6f</td>
</tr>
<tr>
<td>m2</td>
<td>10.0.2.2, 8080, 16, 27, 14, 1e, 4a, 7d</td>
</tr>
</tbody>
</table>

// Proxy LB on cluster service IP to an endpoint on a POD
extern ActionSelector{
  // Construct a selection table for a given ActionProfile.
  ActionSelector(ActionProfile action_profile,
      Hash<2> hash,
      SelectorMode_t mode,
      bit<32> max_group_size,
      bit<32> num_groups);
}

ActionSelector(bit<32> size, Hash<2> hash, SelectorMode_t mode);

// Choose an external hash or PNA hash
Hash<2>selector_hash;

// Pick an entry and apply DNAT
action set_nhop(bit<48> pod_dmac, bit<32> pod_ipv4, bit<9> port) {
  hdr.ipv4.dstAddr = pod_dmac;
  hdr.ipv4.dstPort = pod_ipv4;
}

// Hash to use 5-tuple
// can be tcp, udp, sctp
table 1b{
  key = {
    hdr.ipv4.dstAddr : exact;
    hdr.tcp.dstPort : exact;
    hdr.ipv4.srcAddr : selector;
    hdr.ipv4.dstAddr : selector;
    hdr.ipv4.Tcp.srcPort : selector;
    hdr.ipv4.Tcp.dstPort : selector;
    hdr.ipv4.protocol : selector;
  }
  actions = {
    NoAction;
  }
}
CT P4 Data Plane

- Network Platforms Group
- Open vSwitch

Diagram showing the data plane flow with two firewalls:
- Stateless Firewall for trusted direction
- Stateful Firewall for untrusted direction

Code snippet:
```
action ct_aging_table_hit {
  ... conditions...
}

action ct_aging_table_miss {
  ... conditions...
}
```

Table:
```
table ct_aging_table {
  ... fields...
}
```
L4-L7 P4 Classifier
Opens

• Not every eBPF program can be HW offloaded as is. We are looking at all use cases.

• We would like to get community support in converting some well defined XDP implementations to p4 programs.

• P4 extensions or externs is an option for complete packet transformations like Crypto, checksum, packet replication etc.
Conclusion:
Components for Native Scale out of Container Networking

- Standard Container Control Plane SW
- Enhanced SW Container & Pod Interfaces
- SW to support FNIC/SNIC HW to provide Container Networking offload Through Programmable DP
- SW support for Platform Light-Weight Virtualization

<table>
<thead>
<tr>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard (No Change)</td>
</tr>
<tr>
<td>Need SW Enhancements to benefit from HW</td>
</tr>
</tbody>
</table>
Contacts

P4 Code will be on github soon...

Please email for more info...

Contacts: Anjali Singhai Jain  [anjali.singhai@intel.com](mailto:anjali.singhai@intel.com)

Nupur Jain  [nupur.jain@intel.com](mailto:nupur.jain@intel.com)

Intel Container Networking Team:

<table>
<thead>
<tr>
<th>Anjali Singhai Jain, Nupur Jain, Amritha Nambiar, Pawel Szymanski, Shaopeng He, Phani Burra,</th>
<th>Dan Daly, Yadong Li, Sridhar Samudrala, Kiran Patil, Liang Cunming, Edwin Verplanke</th>
</tr>
</thead>
</table>


eBPF Implementation Characterization – Cilium

Stack trace (NO XDP)

- vmlinux: 615.317s
- nginx: 55.704s
- [Unknown]: 24.101s
- cls_bpf_classify: 21.866s
- __tcf_classify ← tcf_classify_in: 1.173s
- __tcf_classify ← tcf_classify: 1.042s
- [Unknown]: 0.020s
- func@0x7a5bd4 ← func@0x79: 0s
- func@0x7a0570: 0s
- cls_bpf.ko: 23.259s
- ip_tables.ko: 20.728s
- libc-2.13.so: 11.371s
- amplxe-perf: 8.445s
- cls_bpf.ko: 8.339s
- containerd-shim: 8.209s

Stack trace (XDP)

- [Unknown]: 28.626s
- [Outside any known module]: 28.626s
- cls_bpf_classify: 17.400s
- bpf_prog_run_xdp ← 8.194s
- __tcf_classify ← tcf_cl: 1.463s
- __tcf_classify ← tcf_cl: 1.273s
- ice_napi_poll ← napi_poll: 0.236s
- [Unknown]: 0.035s
- func@0x79fe60: 0.015s
- func@0x7a19d5 ← func@0x79: 0.005s
- func@0x79fbc0 ← func@0x79fbe0 ← func@0x7a0570: 0s
- cls_bpf.ko: 22.519s
- ip_tables.ko: 20.728s
- libc-2.13.so: 11.371s
- amplxe-perf: 8.445s
- cls_bpf.ko: 8.339s
- containerd-shim: 8.209s

- XDP vs No XDP, CPU utilization is quite similar
- Benefits of XDP is being able to bypass the kernel stack in case of redirect.
- Redirect to external port requires dedicated HW Redirect TX queue.
- XDP Benefits can be derived from dedicated HW resources.
Cilium chains in iptables

```c
# Generated by iptables-save v1.6.1 on Tue Dec 1 13:22:26 2020

*raw
  :PREUPTOWN ACCEPT [3935660677:8856460677]
  :OUTPUT ACCEPT [19680756:7484100675]
  :RAW_OUTPUT - [0:0]
  :PREUPTOWN_OUTPUT - [0:0]

:RAW_INPUT - [0:0]
  :PREUPTOWN - [0:0]
  :PREUPTOWN_OUTPUT - [0:0]
  :CILIUM - [0:0]
  :POSTROUTING - [0:0]
  :OUTPUT - [0:0]
  :POSTROUTING_OUTPUT - [0:0]
  :CILIUM_OUTPUT - [0:0]

:RAW_FORWARD - [0:0]
  :PREUPTOWN - [0:0]
  :PREUPTOWN_FORWARD - [0:0]
  :CILIUM_FORWARD - [0:0]
  :POSTROUTING - [0:0]
  :OUTPUT - [0:0]
  :POSTROUTING_OUTPUT - [0:0]
  :CILIUM_OUTPUT - [0:0]

:RAW?url - [0:0]
  :PREUPTOWN - [0:0]
  :PREUPTOWN_FORWARD - [0:0]
  :CILIUM_FORWARD - [0:0]
  :POSTROUTING - [0:0]
  :OUTPUT - [0:0]
  :POSTROUTING_OUTPUT - [0:0]
  :CILIUM_OUTPUT - [0:0]

:PREUPTOWN - [0:0]
  :ACCEPT [3935660677:8856460677]
  :DROP [3935660677:8856460677]

:PREUPTOWN_FORWARD - [0:0]
  :ACCEPT [3935660677:8856460677]
  :DROP [3935660677:8856460677]

:POSTROUTING - [0:0]
  :ACCEPT [3935660677:8856460677]
  :DROP [3935660677:8856460677]

:OUTPUT - [0:0]
  :ACCEPT [3935660677:8856460677]
  :DROP [3935660677:8856460677]

:POSTROUTING_OUTPUT - [0:0]
  :ACCEPT [3935660677:8856460677]
  :DROP [3935660677:8856460677]

:PREUPTOWN_OUTPUT - [0:0]
  :ACCEPT [3935660677:8856460677]
  :DROP [3935660677:8856460677]

:POSTROUTING_OUTPUT_OUTPUT - [0:0]
  :ACCEPT [3935660677:8856460677]
  :DROP [3935660677:8856460677]
```