Performance Benchmarking and Tuning for Container Networking on Arm

Trevor Tao,
Jianlin Lv, Jingzhao Ni, Song Zhu

Sep/2020
Agenda

• Introduction
• Container Networking Interfaces (CNIs) on arm
• Benchmarking metrics, environment and tools
• Benchmarking results
• Initial Performance Analysis with perf tools
• Future Work (Provisional)
Introduction
Introduction

What is CNI?
- CNI (Container Network Interface), a Cloud Native Computing Foundation project, consists of a specification and libraries for writing plugins to configure network interfaces in Linux containers, along with a number of supported plugins.
- CNI concerns itself only with network connectivity of containers and removing allocated resources when the container is deleted.
- CNI has a wide range of support and the specification is simple to implement but not the implementation itself for its extensions.
- CNI are the de-facto Kubernetes networking support
- We need to know how they perform on arm platform

Kubernetes Networking Model
- Kubernetes makes opinionated choices about how Pods are networked:
  - all Pods can communicate with all other Pods without using network address translation (NAT).
  - all Nodes can communicate with all Pods without NAT.
  - The IP that a Pod sees itself as is the same IP that others see it as.
Container Networking Interfaces (CNIs) on arm
High Performance CNIs available for Arm Edge Stack

Things now available in **Akraino IEC** Arm edge stack as a ref:

- **Calico**
  - pure IP networking fabric
  - high-level network policy management by iptables
  - Good scalability
  - Support direct(non-overlay) and overlay(IPINIP, VxLAN) network connection
  - Easy deployment
  - Calico-VPP appears

- **Cilium**
  - Linux-Native, API-Aware Networking and Security for Containers
  - Linux eBPF based network policy, load balance and security which is believed to be with incredible performance
  - L3 networking between hosts
  - Good scalability too

- **Contiv-VPP**
  - uses FD.io VPP to provide network connectivity between PODs
  - Native DPDK interface support for phy NIC
  - Native VPP ACL/NAT based network policy and access
  - Good performance but with rather complex configuration
  - Hard to debug

- **OVN-K8s**
  - OVS/OVN-controller based K8s networking solution
  - Rather good performance with OVS inherited
  - Use OVN logical switches/routers to connect Pods and for outside access
  - No OVS-DPDK support now

- **SRIOV**
  - Direct physical interfaces(PF/VFs) support for Pods
  - High performance with direct Linux kernel eth driver or DPDK PMD driver
  - Usually co-work with other CNIs, such as Flannel, Calico by Multus or other glue CNI
  - Need resource description or annotation when do the configuration for CNI and Pod setup

- **Flannel**
  - Widely used and almost easiest deployment for a simple K8s networking
  - Linux network bridge for pod connection and overlay based communication for inter-hosts access
  - Easy to be integrated into other container networking solution, e.g., Cilium
  - No good network policy support

---

**IEC Arm Edge Stack**

**Code Review / iec.git / tree**

- **Fix issue that deploy ciliun on latest image**
  - `iarm-x-x-x`
  - calico
  - tree
  - busy
  - cillum
  - tree
  - busy
  - contiv-vpp
  - tree
  - busy
  - dumm
  - tree
  - busy
  - flannel
  - tree
  - busy
  - multus
  - tree
  - busy
  - ovn-kubernetes
  - tree
  - busy

**Repo:**

https://gerrit.akraino.org/r/admin/repos/iec
CNI Networking Models

**Flannel**

- Backend: IPIP, VXLAN
- Tested version: v0.11.0

**Cilium**

- Backend: VXLAN, Direct Routing (not tested now)
- Tested version: Master branch compiled at 2020-09-09

*Quote from web source*

*Ref. and modified from web source*
CNI Networking Models

Calico

Kubernetes Service Implementation

1. Pure IP networking fabric
2. No encapsulation needed when simple L2 connection available for nodes
3. Easy deployment and debug
4. Supporting Kubernetes Network Policy by iptables
5. Good scalability with BGP based routing

Quote from web source

Tested version: v3.13.2
Benchmarking metrics, environment and tools
Benchmarking metrics, topology and tools

Benchmarking Metrics
• Protocols: TCP, UDP, HTTP(s)
• TCP, UDP Metrics: bandwidth in Mbits/sec, Gbits/sec, round-trip delay in ms
• HTTP(s): Bandwidth in Mbits/sec, Gbits/sec, CPS(Connection per Second), RPS(Request per Second)

Tools:
• IPerf, WRK

Server Platform
Architecture: aarch64
Byte Order: Little Endian
CPU(s): xxx
On-line CPU(s) list: 0-xxx
Thread(s) per core: 4

Network connection:
10Gbps connection by
Ethernet Controller XXV710←----→82599ES 10-Gigabit
SFI/SFP+ Network Connection 10fb

CPU max MHz: 2500.0000
CPU min MHz: 1000.0000
BogoMIPS: 400.00
L1d cache: xxK
L1i cache: xxK
L2 cache: xxxK
L3 cache: xxxxK
NUMA node0 CPU(s): 0-xxx
NUMA node1 CPU(s): xxx-yyy
Benchmarking metrics, environment and tools

**IPerf(v2)** test topology:

- **Test Command:**
  - **Client:**
    - `iperf -c ${SERVER_IP} -t $(time) -i 1 -w 100K -P 4`
  - **Server:**
    - `iperf -s`

**Wrk** (http performance) test topology:

- **Test command:**
  - `wrk -t12 -c1000 -d30s http://$IP/files/$file`
Benchmarking Results
Benchmarking Results of TCP Throughput for CNIs with Different Backends

Node to Pod TCP Performance for IPIP(Calico), IPIP(Flannel), VXLAN(Flannel), VXLAN(Cilium) and Direct Routing(no Tunnel, Calico)
Inter-Hosts 10Gb/s ether connection

Pod to Pod Performance for IPIP(Calico), IPIP(Flannel), VXLAN(Flannel), VXLAN(Cilium) and Direct Routing(no Tunnel, Calico)
Inter Hosts 10Gb/s ether connection

Observation for TCP performance over CNIs

- The performance gap between CNIs are not so explicit when overlay tunnel is used;
- Calico and Flannel show a little bit better performance than Cilium for most MTUs here
- With IPIP/VXLAN overlay tunnel enabled, the larger MTU size, the throughput(BW) performance is better.
- When use direct routing (here by Calico, Cilium also support this mode), the throughput performance is not significantly affected by MTU size.
- The performance of direct routing here by Calico, Cilium also support this mode) is better than IPIP enabled.
- The IPIP tunnel is a little better than VXLAN tunnel
- In general, the node to pod TCP performance is better than that of pod 2 pod which flows one more step (of veth connection to the Linux kernel).

Finally, compared with different scenarios, it proves that IPIP/VXLAN overlay tunnel which are now implemented in the Linux kernel is the key factor which affects the performance of CNIs on arm

Question:
Why the node to pod performance is no better than that of pod to pod case for Cilium?
Initial observation:
- MTU has a rather bigger effect on the performance when accessing large files, but when the accessed file size is small, it has little effect.
- The accessed file size is a major factor to the HTTP performance when there is only a small number of parallel threads.
- When the file size is big enough, the performance can’t be improved much even with bigger MTUs.
HTTP Performance Benchmarking for Calico CNI

Pod2Pod HTTP Performance with Calico non-IPIP Overlay for Cross-Host Communication

Initial observation:
- Almost the same as that of IPIP
- The file size has much more significant performance impact than the MTU
- For file size ≥ 10MB, the MTU has little effect to the final performance
- The performance is much higher than those of IPIP when file size ≥ 100KB (See next page)

Question:
Why for small file size, the performance of smaller MTU is even better than those of large MTUs?

Wrk: thread 5, connections: 10
Pod2Pod HTTP Performance with Calico **IPIP vs non-IPIP** for Cross-Host Communication

**Initial observation:**
- For file size > 10MB, the MTU has little effect to the final performance.
- The performance is much higher than those of IPIP when file size >= 100KB.
- When MTU is small, the performance gap between IPIP and non-IPIP is higher.

Wrk: thread 5, connections: 10
Observation:

- The performance gap is minor when accessing small files.
- For small file size, the host2pod and host2service performance is almost the same, which means the service access (by iptables configured by kube-proxy) is not the bottleneck for HTTP service.
- The performance of non-IPIP is much higher than those of IPIP when file size >= 100KB.
- For large MTU and large file size, the host2pod performance is better than host2svc.
- For non-IPIP, the performance gap between different MTU is not so explicit, so it’s believed the IPIP is actually the bottleneck, which is the same as previous.
HTTP Performance Benchmarking for CNIs with various backends

Pod2Pod HTTP Performance of CNIs for inter-hosts communication

Initial observation:

- For file size ≥ 10MB, the MTU has little effect to the final performance for different CNIs
- When the file size is small, different CNIs has little performance gap
- When the file size is large (≥ 100KB), it shows Calico and Cilium performance much better than Flannel, especially for large MTUs.
- The performance is much higher than those of IPIP when file size ≥ 100KB
- When MTU is small, the performance gap between overlay and non-overlay is higher
Observation:

- For the 3 CNIs, the performance gap is minor when accessing small files.
- As previous, the direct routing (no tunnel) mode shows the best performance compared with any other overlay based approaches.
- For file size >= 100KB, the Calico shows explicitly the best performance over other 2 CNIs.
- Flannel shows the worst host2service performance over other 2 CNIs, even with larger MTUs, for either IPIP tunnel or VXLAN tunnel.
- For large MTU and large file size, Cilium shows similar performance with the Calico CNI.
- For non-IPIP, the performance gap between different MTU is not so explicit, so it’s believed that the tunnel communication is actually the bottleneck, which is the same as previous.
Initial Performance Analysis with perf tools
Initial Performance Analysis with perf tools

Possible performance analysis tools:

- **Ftrace**
- **Perf**
- **DTrace**

The Flamegraphs are got by the following commands:

- `#perf record -e cpu-clock -F 1000 -a -g -C 2 -- sleep 20`
- `#perf script | FlameGraph/stackcollapse-perf.pl > out.perf-folded`
- `#cat out.perf-folded | FlameGraph/flamegraph.pl > perf-kernel.svg`

The Flamegraph script package is got by:

- `git clone https://github.com/brendangregg/FlameGraph.git`
Issues: Performance Analysis for IPerf with Flame Graph

2 flame graphs for w/wo IPIP tunnel of performance test
Summary and Future Work
Brief Summary

With the performance tests for CNIs over arm64 platform, initially we got:

- All 3 CNIs (Calico, Cilium, Flannel) utilize the Linux kernel overlay tunnel implementation to enable its cross-host pod and service communication
- The TCP throughput performance gap between CNIs are not so explicit when overlay tunnel is used;
- For TCP throughput, Calico and Flannel show a little bit better performance than Cilium for most MTUs here
- With IPIP/VXLAN overlay tunnel enabled, the larger MTU size, the throughput (BW) performance is better.
- The overlay tunnel approaches (IPIP, VXLAN) actually affects the performance either TCP or HTTP performance much compared with direct routing;
- For HTTP performance, the Calico and Cilium shows much better performance over Flannel CNI
Future Work (Provisional)

- Performance testing for supported senior features of CNIs
  - Kube-proxy replacement with eBPF for Cilium CNI
  - Encryption added for pod2pod communication of Cilium CNI
  - eBPF introduced for Calico CNI
- HTTP performance testing with network policy configured (Cilium, Calico)
- Further performance trace, analysis and optimization for known performance issues
- Performance testing for other CNIs on arm: Ovn-Kubernetes, Contiv/VPP
- More backend types testing for Cilium, Calico or other CNIs
- Compare with other platform (x86_64, ...)
- Investigation on the performance differences between CNIs
- ...

© 2020 Arm Limited (or its affiliates)
Thank You
Danke
Merci
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה