Benchmarking Various CNI Plugins

Giri Kuncoro & Vijay Dhama from GOJEK
Agenda

- Overview of Various CNI Plugins
- Experiments
  - Goals
  - Environment
  - Results
- Takeaways
Overview of Various CNI Plugins
Kubernetes Network Model

- All containers communicate without NAT
- All nodes communicate with containers without NAT
- Container sees its own IP as others see it

Kubernetes doesn’t provide default network implementation, it leaves it to third party tools

source: https://kubernetes.io/docs/concepts/cluster-administration/networking/#kubernetes-model
What CNI do?

Connectivity

Reachability
CNI Plugins

**Project Calico** - a layer 3 virtual network

**Weave** - a multi-host Docker network

**Contiv Networking** - policy networking for various use cases

**SR-IOV**

**Cilium** - BPF & XDP for containers

**Infoblox** - enterprise IP address management for containers

**Multus** - a Multi plugin

**Romana** - Layer 3 CNI plugin supporting network policy for Kubernetes

**CNI-Genie** - generic CNI network plugin

**Nuage CNI** - Nuage Networks SDN plugin for network policy kubernetes support

**Silk** - a CNI plugin designed for Cloud Foundry

**Linen** - a CNI plugin designed for overlay networks with Open vSwitch and fit in SDN/OpenFlow network environment

**Vhostuser** - a Dataplane network plugin - Supports OVS-DPDK & VPP

**Amazon ECS CNI Plugins** - a collection of CNI Plugins to configure containers with Amazon EC2 elastic network interfaces (ENIs)

**Bonding CNI** - a Link aggregating plugin to address failover and high availability network

**ovn-kubernetes** - an container network plugin built on Open vSwitch (OVS) and Open Virtual Networking (OVN) with support for both Linux and Windows

**Juniper Contrail / TungstenFabric** - Provides overlay SDN solution, delivering multicloud networking, hybrid cloud networking, simultaneous overlay-underlay support, network policy enforcement, network isolation, service chaining and flexible load balancing

**Knitter** - a CNI plugin supporting multiple networking for Kubernetes

source: https://github.com/containernetworking/cni#3rd-party-plugins
So many CNI plugins to test, limit scope to:

- Flannel
- Calico
- Weave
- Cilium
- Kube-Router
- AWS CNI
- Kopeio
- Romana
Flannel

Simple way to configure L3 network fabric with VXLAN as default

source: https://github.com/coreos/flannel
Calico

Pure L3 approach which enables unencapsulated networks and BGP peering

source: https://projectcalico.org
Support overlay network with different cloud network config

source: https://www.weave.works/docs
Cilium

Based on Linux kernel technology called BPF

source: https://cilium.readthedocs.io
Kube-Router

Built on standard Linux networking toolset:
ipset, iptables, IPVS, LVS

source: https://github.com/cloudnativelabs/kube-router
Using AWS ENI interface for pod networking

source: https://github.com/aws/amazon-vpc-cni-k8s
Kopeio

Simple VXLAN, but also support L2 with GRE and IPSEC

source: https://github.com/kopeio/networking
Use standard L3, distributed routes with BGP or OSPF
Experiments
Goals

- Lowest latency and highest throughput
- Different protocols and various packet sizes
- CPU consumption and launch time
- Kubernetes network policies
Environment

- 8 Kubernetes clusters with different CNIs
- 2 nodes cluster with **m4.xlarge** type by Amazon AWS EC2 with **Debian 9, kernel 4.9**
- **Kubernetes v1.10.9** with **Kops**
● All CNI plugins deployed with default config in Kops
  ○ No tuning or custom configuration
  ○ Flannel v0.10.0
  ○ Calico v2.6.7
  ○ Weave v2.4.0
  ○ Cilium v1.0
  ○ Kube-Router v0.1.0
  ○ AWS CNI v1.0.0
  ○ Kopeio v1.0.20180319
  ○ Romana v2.0.2
Tools

- **Sockperf (v3.5.0)**
  - Util over socket API for latency/throughput measurement

- **Netperf (v2.6.0)**
  - Unidirectional throughput and end-to-end latency measurement

- **Tool from PaniNetworks**
  - Generate HTTP workloads and measure response
Experiment #1
Throughput & Latency
Steps

- Sockperf client pod in Node A
- Sockperf server pod in Node B

- 256 bytes for TCP throughput test
- 16 bytes for TCP latency test
TCP throughput

![Graph showing TCP throughput comparison with baseline]
TCP throughput

Baseline

Relative to host network

flannel
weave
kopeio
romana
TCP throughput

![Graph showing TCP throughput](image_url)

Baseline

Relative to host network

- calico
- cilium
- kube-router
- aws-cni

**Note:** The image shows a bar chart comparing different network solutions in terms of TCP throughput relative to the host network.
TCP latency

Relative to host network

- aws-cni
- romana

Baseline
TCP latency

![Graph showing TCP latency baseline and kopeio comparison.]

The graph illustrates the relative latency compared to the host network. The baseline is indicated by a dashed line at 1.00. The kopeio performance is shown in pink, highlighting a significant increase in latency compared to the baseline.
Experiment #2
Protocol & Packet Sizes
Steps

- Sockperf client pod in Node A
- Sockperf server pod in Node B
- Measure TCP and UDP throughput from 16 to 256 bytes
TCP throughput vs packet sizes

- host-net
- flannel
- calico
- cilium
- aws-cni
- kopeio
- weave
- kube-router
- romana

TCP throughput (MBps)

Packet size (Byte)
TCP throughput vs packet sizes

Packet size (Byte)

TCP throughput (MBps)

- flannel
- weave
- kopeio
- romana
TCP throughput vs packet sizes

- calico
- cilium
- aws-cni
- kube-router

TCP throughput (MBps) vs Packet size (Byte)
UDP throughput vs packet sizes

Packet size (Byte)
UDP throughput vs packet sizes

Packet size (Byte) vs UDP throughput (MBps)

- aws-cni
- romana
Steps

- Netperf client pod in Node A
- Netperf server pod in Node B
- Netperf UDP_RR to measure CPU utilization
- Time spent in user space, kernel space, and waiting for I/O
CPU Overhead

The diagram illustrates the CPU overhead relative to the baseline, with various network plugins such as host-net, calico, flannel, cilium, weave, aws-cni, kopeio, kube-router, and romana. The y-axis represents the relative overhead compared to the baseline, while the x-axis shows different metrics such as CPU Util(c), CPU Util(s), CPU Time(c), and CPU Time(s).
CPU Overhead

![Chart showing CPU Overhead](chart.png)

- **calico**
- **aws-cni**
- **kopeio**
- **kube-router**
- **romana**

Relative to host network

- **CPU Util(c)**
- **CPU Util(s)**
- **CPU Time(c)**
- **CPU Time(s)**

Baseline
CPU Overhead

![Chart showing CPU Overhead comparison between weave and cilium across different metrics: CPU Utilization (c), CPU Utilization (s), CPU Time (c), CPU Time (s). The baseline is indicated by a dashed line.](chart.png)
Experiment #4
Network Launch Time
- Deploy Nginx pod in Node A
- Watch kubelet events on container create
- Stopwatch until pinging from Node B
Network launch time
Network launch time
Network launch time

Launch time (sec)

Weave
Calico
AWS-CNI
Romana

Baseline
Network launch time

- cilium
- kube-router
- kopeio

Launch time (sec)

Baseline
Experiment #5
Network Policies
Steps

- Client pod in Node A
- Server pod in Node B
Steps

- Client pod sends 2,000 HTTP requests
- Varied response size from 1KB to 100KB
- Disabled persistent connection
Steps

- Varied network policy from 0 to 200 policies

![Diagram showing network setup with nodes and pods.]

- Node A (Client):
  - pod
  - 2,000 requests

- Node B (Server):
  - pod
  - Default deny
  - Allow ingress to port A
  - ... Allow ingress to port Z
  - Allow ingress to port 80

- 1KB to 100KB responses
Network Policy (Calico)
Network Policy (Weave)

- Graph showing response time in milliseconds (ms) against the number of policies for different policy counts (100K, 10K, 1K).

- The x-axis represents the number of policies ranging from 0 to 200.

- The y-axis represents response time ranging from 0.6 to 1.2 ms.

- The graph indicates a trend where response time remains relatively stable as the number of policies increases.
Network Policy (Cilium)
Network Policy (Kube-Router)
Takeaways
Takeaways

Very challenging to pick appropriate CNI plugins
Tradeoffs between performance, security, isolation
In general, with default config, **Flannel, Weave, Romana** achieves better network performance than the others.
Takeaways

Most CNI plugins get much larger throughput to TCP than UDP workloads.
Takeaways

Most CNI plugins scaled more in UDP with considerable loss
Takeaways

Most CNI plugins introduce **CPU overhead** with **Weave** and **Cilium** being largest.
Takeaways

Most CNI plugins add *reasonable delay* to launch containers, except Flannel.
Most CNI plugins introduce **small delay** as number of policies increases.
Need more comprehensive analysis on the experiment results
Perform specific configuration to allow better performance on some CNI Plugins
More types of experiments on concurrent requests and workload interference
References

An Analysis and Empirical Study of Container Networks, Kun Suo et.al
http://ranger.uta.edu/~jrao/papers/INFOCOM18.pdf

Comparison of Networking Solutions for Kubernetes, Machine Zone
http://machinezone.github.io/research/networking-solutions-for-kubernetes

High Performance Network Policies in Kubernetes Clusters, Pani Networks
Acknowledgements

- Ajey Gore, GO-JEK
- Gaetano Borgione, VMware
- Ali Khayam, Amazon Web Services
- Samudra Bekti, Cisco
谢谢

Giri Kuncoro, giri.kuncoro@go-jek.com
Vijay Dhama, vijay.dhama@go-jek.com