Kernel-based forwarding plane for Network Service Mesh

Radoslav Dimitrov, Open Source Software Engineer, VMware

@radoslav_dimitr @vmwopensource
Agenda

• Introduction to Network Service Mesh
• What is a Kernel-based forwarding plane?
• Demo - Network Service Mesh on ARM64
Network Service Mesh: Introduction

- Clients – consume Network Services
- Endpoints – implement Network Services
- Wires – responsible for the connections, payload agnostic

Public Cloud

Network Service: “secure-intranet-connectivity”

Application → Firewall → VPN Gateway → Intranet
Network Service Mesh: Introduction

- Service mesh for L2/L3 payloads
- On-demand, dynamic, negotiated connections
- Workload-To-Workload granular level of connectivity
- Loosely coupled heterogeneous network configurations
- gRPC API to publish and consume Network Services
- Without changes to Kubernetes
- Works with any CNI
Network Service Mesh: K8s and CNI

- Endpoints announce themselves to NSM
- Client requests a Network Service
- NSM takes care of creating the necessary connections
- NSM lives peacefully with the existing CNI network (eth0)
Network Service Mesh: Connections

Local connections

Client ➔ Request ➔ NSM Manager ➔ Register ➔ Firewall

- secure-intranet-connectivity

Node 1
Remote connections

Client → NSM Manager → Forwarding Plane

Node 1

Request

Tunnel

Node 2

Request

Secure-intranet-connectivity

Firewall Node 2

nsManager

kubernetes

secure-intranet-connectivity

Remote connections
Network Service Mesh: Forwarding plane

- VPP - default forwarding plane
  - High-performance packet processing stack
    - DPDK, hugepages, memif
  - Modular and flexible
  - Rich of features and supported protocols
  - An FD.io project

- Kernel-forwarder
  - Not that performance oriented
  - Instead, it aims to be a simple, self-contained and lightweight solution
  - Based on network tools available on every Linux host
What is a Kernel-based forwarding plane?

- **What is it?**
  - Pure Kernel-based implementation using *netlink*
  - No external dependencies
  - Lightweight and portable

- **How it helps?**
  - Pave the way for other implementations
  - Enable support for multiple *simultaneous* forwarding planes
    - “One forwarding plane cannot rule them all”
    - For example, a client requests a specific type of connection (i.e., SRIOV VF), then we might depend on a specific forwarding plane coupled for a given NIC
Kernel-forwarder: Overview

1. Client sends a request
2. NSMgr generates a **Cross-connect request** passed to the forwarding plane
3. Kernel-forwarder creates the connection using the **netlink** interface
4. Both Pods get their connection interfaces attached
Local connections

1. Create a VETH pair - host ns
2. Inject the 1\textsuperscript{st} - client ns
3. Inject the 2\textsuperscript{nd} - endpoint ns
4. Setup each interface – set IP, name, etc.
5. Connection completed!
Remote connections

1. Create a VxLAN decapsulated interface – **host ns**
2. Inject it – **client/endpoint ns**
3. Setup the interface – set IP, name, etc
4. Connection completed!
<table>
<thead>
<tr>
<th>NAMESPACE</th>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>kube-system</td>
<td>coredns-5c98db65d4-h8crh</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>16m</td>
</tr>
<tr>
<td>kube-system</td>
<td>coredns-5c98db65d4-zx8r8x</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>16m</td>
</tr>
<tr>
<td>kube-system</td>
<td>etcd-rock64</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>15m</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-apiserver-rock64</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>15m</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-controller-manager-rock64</td>
<td>1/1</td>
<td>Running</td>
<td>1</td>
<td>15m</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-proxy-hgcmd</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>14m</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-proxy-zwjsjm</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>16m</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-scheduler-rock64</td>
<td>1/2</td>
<td>Running</td>
<td>0</td>
<td>15m</td>
</tr>
<tr>
<td>kube-system</td>
<td>weave-net-4prph</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>13m</td>
</tr>
<tr>
<td>kube-system</td>
<td>weave-net-165gr</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>13m</td>
</tr>
</tbody>
</table>

```
rock64@rock64:~ watch -n 0.5 'kubectl logs -n nsm-system nsm-kernel-forwarder -l tail -n 25'
```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.4.132-1072-rockchip-ayufan-ga1d27dba5a2e aarch64)

* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

System information as of Sat Sep 21 12:39:07 UTC 2019

System load: 0.89
Users logged in: 1
Usage of /: 61.9% of 28.31GB
Memory usage: 25%
Swap usage: 0%
Processes: 211

IP address for eth0: 192.168.0.102
IP address for docker0: 172.17.0.1
IP address for weave: 10.40.0.0

* Congrats to the Kubernetes community on 1.16 beta 1! Now available in MicroK8s for evaluation and testing, with upgrades to RC and GA

Last login: Sat Sep 21 12:38:31 2019 from 192.168.0.100
rock64@rock64:~$
Kernel-forwarder: Outro

• Overview:
  • Self-contained, lightweight and portable
  • Multi-arch support
  • VxLAN for remote connectivity
  • Custom routes and neighbors support
  • Metrics monitoring support
  • Paves the way for other forwarding plane implementations

• Ideas to consider:
  • Add more remote protocols
    • GRE, GENEVE, SR, MPLS
  • Add fancy interfaces
    • VXCAN, RDMA
  • SRIOV
  • eBPF and XDP
Thank you!

dimitrovr@vmware.com

https://github.com/networkservicemesh

@radoslav_dimitr @vmwopensource
Thank you!

dimitrovr@vmware.com

https://github.com/networkservicemesh

@radoslav_dimitr @vmwopensource