FD.io VPP & Ligato Use Cases
Contiv-VPP CNI plugin for Kubernetes
IPSEC VPN gateway
fd.io VPP – The Universal Dataplane

• Project at Linux Foundation
  • Multi-party
  • Multi-project

• Software Dataplane
  • High throughput
  • Low Latency
  • Feature Rich
  • Resource Efficient
  • Bare Metal/VM/Container
  • Multi-platform

• fd.io Scope:
  • Network IO - NIC/vNIC <-> cores/threads
  • Packet Processing – Classify/Transform/Prioritise/Forward/Terminate
  • Dataplane Management Agents - Control Plane

Bare Metal/VM/Container

Dataplane Management Agent
Packet Processing
Network I/O
fd.io VPP - In the Overall Stack

- Application Layer/App Server
- Orchestration
- Network Controller
- Data Plane Services
  - Dataplane Management Agent
  - Packet Processing
  - Network I/O
- Operating System
- Hardware
fd.io VPP - Projects

Dataplane Management Agent
- Honeycomb
- hc2vpp
- GoVPP

Packet Processing
- NSH_SFC
- ONE
- TLDK
- CICN
- odp4vpp
- P4VPP
- VPP

Network IO
- deb_dpdk
- rpm_dpdk

Testing/Support
- CSIT
- puppet-fdio
- TRex
- Sandbox
Packet Processing Software Platform

- High performance
- Linux user space
- Runs on compute CPUs: Intel, ARM
  - And “knows” how to run them well!

Shipping at volume in server & embedded products

fd.io VPP – Vector Packet Processing
Compute Optimised SW Network Platform
Packet processing is decomposed into a directed graph of nodes … packets move through graph nodes in vectors … graph nodes are optimised to fit inside the instruction cache …

Makes use of modern Intel® Xeon® Processor micro-architectures.
Instruction cache & data cache always hot ➔ Minimised memory latency and usage.

Each graph node implements a “micro-NF”, a “micro-NetworkFunction” processing packets.
Packet Processing

Graph Node

Input Graph Node

Graph Node

Packet Processing

Packet

Vector of n packets

Packet

Packet Processing

Graph

Vector of n packets

Input Graph Node

Graph Node

Packet

Packet Processing

Graph
fd.io VPP – Architecture

Splitting the Vector

Packet Processing Graph

Graph Node

Input Graph Node

Packet

Vector of n packets

dpdk-input ➔ vhost-user-input ➔ ... ➔ af-packet-input ➔ ethernet-input ➔ ip6-input ➔ ip4-input ➔ mpls-input ➔ arp-input ➔ ip6-lookup ➔ ip4-lookup ➔ ip6-rewrite ➔ ip6-local ➔ ip4-local ➔ ip4-rewrite
fd.io VPP - Architecture

Plugins

Hardware Plugin

hw-accel-input

dpdk-input

vhost-user-input

af-packet-input

Packet Processing Graph

Input Graph Node

Graph Node

Packet

Vector of n packets

0 1 2 3 ... n

Packet

Plugins are:
First class citizens
That can:
Add graph nodes
Add API
Rearrange graph

Can be built independently of VPP source tree

Vector of n packets

Skip S/W
nodes where work is
done by hardware already

Plugin
/usr/lib/vpp_plugins/foo.so

Plugin

custom-1

custom-2

ip6-replace

ip6-local

ip4-local

ip4-replace

ip6-lookup

ip4-lookup

mpls-input

arp-input

ip4-input

ip6-input

ethernet-input

ethernet-input

...
Cloud-Native Network Functions with VPP & Ligato

- **cncf.io**: "Cloud native computing uses an open source software stack to deploy **applications as microservices**, packaging each part into its **own container**, and dynamically **orchestrating** those containers to optimize resource utilization."

- **CNFs**: Splitting of network functions into collection (chain) of loosely coupled services, deployed as containers, interconnected with fast data plane links

- **VPP**: data plane part for building CNFs (memifs for interconnecting)

- **Ligato**: control plane part for building CNFs (VPP agent) and CNF chaining (SFC Controller)
Service Function Chaining with CNFs & VPP

Logical Representation

Placement (K8s) → Rendering → Topology

Physical Representation

Ingress Network Ingress Classifier
Egress Network Egress Classifier

Overlay Tunnel

Ingress Router → NF₁ → NF₂ → NF₃ → Egress Router

Host

Server

github.com/ligato/sfc-controller
networkservicemesh.io
Ligato VPP Agent
(Development Platform for VPP-based / Non-VPP-based CNFs)

VPP-based CNFs require a cloud-native management agent for VPP:

- NETCONF/RESTCONF + YANG does not fit to cloud-native very well

- **Protobuf API** is used instead; via any transport:
  - gRPC / REST (for RPC-based configuration / state data retrieval)
  - Key-value datastores: ETCD, Redis, Bolt DB, ... (asynchronous configuration)
  - Message brokers: Kafka, (RabbitMQ, NATS, ...)

- Needs to be pluggable into existing cloud-native infra (k8s, Docker)
  - Implemented in **Go** (as well as Docker & k8s)
  - The agent can be used as a library from other Go applications (Contiv-VPP)

- Needs to be compact and low footprint
  - Packaged into **Docker container images** together with VPP
  - Can be running in multiple containers on the same host
  - Need to have fast startup times (containers can die and restart frequently)
  - VPP Agent is written in Go – it is a single executable (binary)

[github.com/ligato/vpp-agent](github.com/ligato/vpp-agent)
Ligato VPP Agent
(Development Platform for VPP-based / Non-VPP-based CNFs)

**VPP:**
- Binary API – shared memory / unix domain socket

**GoVPP:**
- Golang wrappers over binary APIs (generator)
- Go struct to binary API marshalling & unmarshalling, communication with VPP
- Still binary API
  - low-level (numeric references to interfaces, byte ordering issues, bit flags, etc.)
  - API message ordering / dependency issues
  - API versioning issues (API needs to match with VPP version)

**VPP Agent:**
- Provides protobuf-modelled NB API
  - more high-level, uses labels (names) for referencing
  - does not change frequently, allows backward-compatible API changes
  - various transports for the same API: RPC (gRPC, REST), key-value store (ETCD, Redis), ...
- KVScheduler
  - transaction-based configuration processing using graph processing engine (addresses the ordering issue)
  - error state handling: handles restarts, retries on error, auto-healing (resync)
- Modular & extendable – VPP plugins, Linux plugin
- Packaged as a Docker container together with compatible VPP (no versioning issues)
Ligato VPP Agent
(Development Platform for VPP-based / Non-VPP-based CNFs)
Contiv-VPP
(Container Network Interface Plugin for Kubernetes)

Ligato VPP Agent integrated into Kubernetes ecosystem:

• Provides k8s CNI (Container Network Interface) functionality
  • interconnects the PODs in the cluster (TAP interfaces)
  • uses VXLAN tunnels for node interconnection or no overlay mode
  • provides kube-proxy functionality (load-balancing + NAT) on VPP
  • Implements k8s policies as ACLs on VPP

• At the same time, still exposes the same APIs as the Ligato VPP Agent
  • uses Ligato VPP Agent as a library to program VPP
  • can be easily extended with extra configuration – e.g. to create additional memif interfaces to CNF PODs

github.com/contiv/vpp
Contiv-VPP
(Container Network Interface Plugin for Kubernetes)

github.com/contiv/vpp
Contiv-VPP
(Container Network Interface Plugin for Kubernetes)
Contiv-VPP + CNFs

Data Plane Network X

Data Plane Network Y

Physical Device

Physical Device

Physical Device

Physical Device

Cloud tools (DB, Mbus, ...)

Contiv Control Plane

Etcd

Kubernetes

SFC Controller

Host

Host

CNF

Agent

CNF

VPP

Cloud app

Cloud app

Contiv-VPP

Agent

VRF X

VRF Y

Cloud VRF

DPDK

NICs

Contiv-VPP

Agent

VRF X

VRF Y

Cloud VRF

DPDK

NICs
k8s Services & Load Balancing with Contiv-VPP

github.com/contiv/vpp
k8s Services & Load Balancing with Contiv-VPP

vagrant@k8s-master:~$ kubectl get svc nginx
NAME      TYPE           CLUSTER-IP      EXTERNAL-IP       PORT(S)        AGE
nginx     LoadBalancer   10.107.167.81   192.168.16.150   80:32676/TCP   1m

vagrant@k8s-master:~$ kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP         NODE          NOMINATED NODE
nginx-bnh78   1/1       Running   0          1m        10.1.3.2   k8s-worker2  <none>
nginx-rghrf   1/1       Running   0          1m        10.1.4.2   k8s-worker3  <none>
nginx-w57kx   1/1       Running   0          1m        10.1.2.2   k8s-worker1  <none>

MetalLB (https://metallb.universe.tf) used for external load balancing
k8s Services & Load Balancing with Contiv-VPP

vagrant@k8s-worker1:~$ sudo vppctl

vpp# sh inter addr
GigabitEthernet0/8/0 (up):
  L3 192.168.16.2/24
local0 (dn):
  loop0 (up):
    L3 10.1.2.1/24 ip4 table-id 1 fib-idx 1
loop1 (up):
  L2 bridge bd-id 1 idx 1 shg 1 bvi
  L3 192.168.30.2/24 ip4 table-id 1 fib-idx 1
tap0 (up):
  L3 172.30.2.1/24
tap1 (up):
  unnumbered, use loop0
  L3 10.1.2.1/24 ip4 table-id 1 fib-idx 1
vxlan_tunnel10 (up):
  L2 bridge bd-id 1 idx 1 shg 1
vxlan_tunnel11 (up):
  L2 bridge bd-id 1 idx 1 shg 1
vxlan_tunnel2 (up):
  L2 Bridge bd-id 1 idx 1 shg 1

vagrant@k8s-worker1:~$ sudo vppctl

vpp# sh nat44 static mappings
tcp external 10.107.167.81:80 self-twice-nat out2in-only
  local 10.1.2.2:80 vrf 1 probability 1
  local 10.1.3.2:80 vrf 1 probability 1
  local 10.1.4.2:80 vrf 1 probability 1
tcp external 192.168.16.150:80 twice-nat out2in-only
  local 10.1.2.2:80 vrf 1 probability 1
  local 10.1.3.2:80 vrf 1 probability 1
  local 10.1.4.2:80 vrf 1 probability 1

vagrant@k8s-worker1:~$ sudo vppctl

vpp# sh ip arp
Proxy arps enabled for:
  Fib_index 0  192.168.16.150 - 192.168.16.150
k8s Services & Load Balancing with Contiv-VPP

L2 load-balancing (failover)  
externalTrafficPolicy: Cluster

BGP load-balancing  
externalTrafficPolicy: Local

github.com/contiv/vpp
IPSEC in fd.io VPP

Forwarding Plane:
• ESP tunnel/transport modes
• IPv4/IPv6
• SHA up to 512-256 and AES-CBC 128/192/256

Control Plane:
• IKEv2 Initiator and Responder

Tested/Documented Interop with other stacks (e.g. StrongSwan)
Migration from VM/StrongSwan to Contiv/Ligato VNFs

- Server running a VM with StrongSwan and Linux kernel
- Server running StrongSwan with a plugin to VPP for higher IPSEC pps throughput
- Ligato CNF running StrongSwan, a Ligato agent, and a VPP for IPSEC data plane - with a memIf to the Ligato vSwitch
- Kubernetes is optional
- Full K8s/Contiv-vpp integration with ligato CNF, dynamic re-wiring of CNFs using sfc-controller based on k8s placement policies
Multiple StrongSwan CNFs may be deployed on a host. K8s handles CNF placement/lifecycle.

N/B VXLAN tunnel between the host’s vSwitch and the north bound router towards the secure apps

S/B VXLAN tunnel between the host’s vSwitch and the south bound router towards the remote IPSEC clients

sswan-controller configures memifs and VXLAN tunnels, sends BGP updates via sfc-controller plugins, and sends security config to the StrongSwan CNFs
Backup
Network Micro-Service Use Case: Service Function Chaining with Cloud-Native NFs