vpp-firstcut Documentation

Release 0.1

John DeNisco

May 30, 2018

Contents

FD.io (Fast data - Input/Output) is a collection of several projects and libraries to amplify the transformation to support flexible, programmable and composable services on a generic hardware platform. FD.io offers the Software Defined Infrastructure developer community a landing site with multiple projects fostering innovations in software-based packet processing towards the creation of high-throughput, low-latency and resource-efficient IO services suitable to many architectures (x86, ARM, and PowerPC) and deployment environments (bare metal, VM, container).

A key component is the Vector Packet Processing (VPP) library donated by Cisco.

CHAPTER 1

Concepts

1.1 What is VPP?

1.1.1 Introduction

The VPP platform is an extensible framework that provides out-of-the-box production quality switch/router functionality. It is the open source version of Cisco's Vector Packet Processing (VPP) technology: a high performance, packet-processing stack that can run on commodity CPUs.

The benefits of this implementation of VPP are its high performance, proven technology, its modularity and flexibility, and rich feature set.

The VPP technology is based on proven technology that has helped ship over $1 Billion of Cisco products. It is a modular design. The framework allows anyone to "plug in" new graph nodes without the need to change core/kernel code.

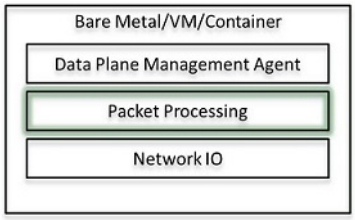

Fig. 1: Packet Processing Layer in High Level Overview of Networking Stack

1.1.2 Why is it called vector processing?

As the name implies, VPP uses vector processing as opposed to scalar processing. Scalar packet processing refers to the processing of one packet at a time. That older, traditional approach entails processing an interrupt, and traversing the call stack (a calls b calls c... return return return from the nested calls... then return from Interrupt). That process then does one of 3 things: punt, drop, or rewrite/forward the packet.

The problem with that traditional scalar packet processing is:

- thrashing occurs in the I-cache
- each packet incurs an identical set of I-cache misses
- no workaround to the above except to provide larger caches

By contrast, vector processing processes more than one packet at a time.

One of the benefits of the vector approach is that it fixes the I-cache thrashing problem. It also mitigates the dependent read latency problem (pre-fetching eliminates the latency).

This approach fixes the issues related to stack depth / D-cache misses on stack addresses. It improves "circuit time". The "circuit" is the cycle of grabbing all available packets from the device RX ring, forming a "frame" (vector) that consists of packet indices in RX order, running the packets through a directed graph of nodes, and returning to the RX ring. As processing of packets continues, the circuit time reaches a stable equilibrium based on the offered load.

As the vector size increases, processing cost per packet decreases because you are amortizing the I-cache misses over a larger N.

1.1.3 Modular, Flexible, and Extensible

The VPP platform is built on a 'packet processing graph'. This modular approach means that anyone can 'plugin' new graph nodes. This makes extensibility rather simple, and it means that plugins can be customized for specific purposes.

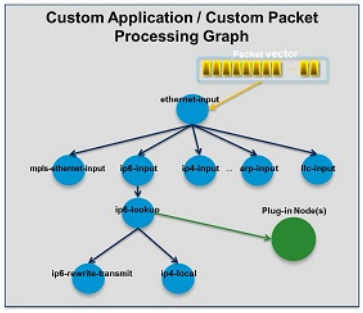

Fig. 2: Custom Packet Processing Graph

How does the plugin come into play? At runtime, the VPP platform grabs all available packets from RX rings to form a vector of packets. A packet processing graph is applied, node by node (including plugins) to the entire packet vector. Graph nodes are small and modular. Graph nodes are loosely coupled. This makes it easy to introduce new graph nodes. It also makes it relatively easy to rewire existing graph nodes.

A plugin can introduce new graph nodes or rearrange the packet processing graph. You can also build a plugin independently of the VPP source tree - which means you can treat it as an independent component. A plugin can be installed by adding it to a plugin directory.

The VPP platform can be used to build any kind of packet processing application. It can be used as the basis for a Load Balancer, a Firewall, an IDS, or a Host Stack. You could also create a combination of applications. For example, you could add load balancing to a vSwitch.

The engine runs in pure userspace. This means that plugins do not require changing core code - you can extend the capabilities of the packet processing engine without the need to change code running at the kernel level. Through the creation of a plugin, anyone can extend functionality with:

- New custom graph nodes

- Rearrangement of graph nodes
- New low level APIs

1.1.4 Feature Rich

The full suite of graph nodes allows a wide variety of network appliance workloads to be built. At a high level, the platform provides:

- Fast lookup tables for routes, bridge entries
- Arbitrary n-tuple classifiers
- Out of the box production quality switch/router functionality

The following is a summary of the features the VPP platform provides:

List of Features

IPv4/IPv6

- 14+ MPPS, single core
- Multimillion entry FIBs
- Input Checks
  - Source RPF
  - TTL expiration
  - header checksum
  - L2 length < IP length
  - ARP resolution/snooping
  - ARP proxy
- Thousands of VRFs
  - Controlled cross-VRF lookups
- Multipath – ECMP and Unequal Cost
- Multiple million Classifiers - Arbitrary N-tuple
- VLAN Support – Single/Double tag

IPv4

- GRE, MPLS-GRE, NSH-GRE,
- VXLAN
- IPSEC
- DHCP client/proxy

IPv6

- Neighbor discovery
- Router Advertisement
- DHCPv6 Proxy
- L2TPv3
- Segment Routing
- MAP/LW46 – IPv4aas
- iOAM

MPLS

- MPLS-o-Ethernet – Deep label stacks supported

L2

- VLAN Support
  - Single/ Double tag
  - L2 forwarding with EFP/BridgeDomain concepts
- VTR – push/pop/Translate (1:1,1:2, 2:1,2:2)
- Mac Learning – default limit of 50k addresses
- Bridging – Split-horizon group support/EFP Filtering
- Proxy Arp
- Arp termination
- IRB – BVI Support with RouterMac assignment
- Flooding
- Input ACLs
- Interface cross-connect

1.1.5 Example Use Case: VPP as a vSwitch/vRouter

One of the use cases for the VPP platform is to implement it as a virtual switch or router. The following section describes examples of possible implementations that can be created with the VPP platform. For more in depth descriptions about other possible use cases, see the list of

Note: jadfix todo Link to the Use Cases

You can use the VPP platform to create out-of-the-box virtual switches (vSwitch) and virtual routers (vRouter). The VPP platform allows you to manage certain functions and configurations of these application through a command-line interface (CLI).

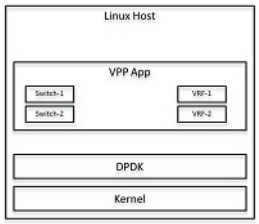

Fig. 3: Figure: Linux host as a vSwitch

Some of the functionality that a switching application can create includes:

- Bridge Domains
- Ports (including tunnel ports)
- Connect ports to bridge domains
- Program ARP termination

Some of the functionality that a routing application can create includes:

- Virtual Routing and Forwarding (VRF) tables (in the thousands)
- Routes (in the millions)

1.1.6 Local Programmability

One approach is to implement a VPP application to communicate with an external application within a local environment (Linux host or container). The communication would occur through a low level API. This approach offers a complete, feature rich solution that is simple yet high performance. For example, it is reasonable to expect performance yields of 500k routes/second.

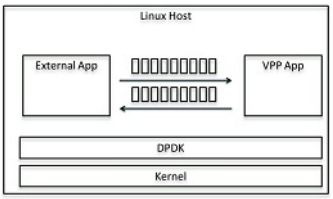

Fig. 4: VPP Communication Through Low Level API

This approach takes advantage of using a shared memory/message queue. The implementation is on a local on a box or container. All CLI tasks can be done through API calls.

The current implementation of the VPP platform generates Low Level Bindings for C clients and for Java clients. It's possible for future support to be provided for bindings for other programming languages.

1.1.7 Remote Programmability

Another approach is to use a Data Plane Management Agent through a High Level API. As shown in the figure, a Data Plane Management Agent can speak through a low level API to the VPP App (engine). This can run locally in a box (or VM or container). The box (or container) would expose higher level APIs through some form of binding.

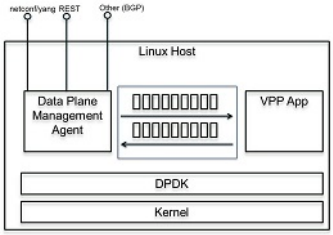

Fig. 5: Figure: API Through Data Plane Management Agent

This is a particularly flexible approach because the VPP platform does not force a particular Data Plane Management Agent. Furthermore, the VPP platform does not restrict communication to only *one* high level API. Anybody can bring a Data Plane Management Agent. This allows you to match the high level API/Data Plane Management Agent and implementation to the specific needs of the VPP app.

1.1.8 Sample Data Plane Management Agent

One example of using a high hevel API is to implement the VPP platform as an app on a box that is running a local ODL instance (Honeycomb). You could use a low level API over generated Java Bindings to talk to the VPP App, and expose Yang Models over netconf/restconf NB.

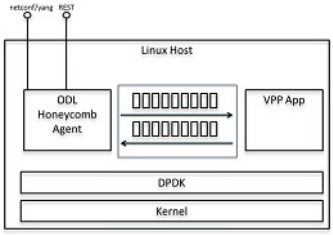

Fig. 6: VPP Using ODL Honeycomb as a Data Plane Management Agent

This would be one way to implement Bridge Domains.

1.1.9 Primary Characteristics Of VPP

Improved fault-tolerance and ISSU

Improved fault-tolerance and ISSU when compared to running similar packet processing in the kernel:

- crashes seldom require more than a process restart
- software updates do not require system reboots
- development environment is easier to use and perform debug than similar kernel code
- user-space debug tools (gdb, valgrind, wireshark)
- leverages widely-available kernel modules (uio, igb_uio): DMA-safe memory

Runs as a Linux user-space process:

- same image works in a VM, in a Linux container, or over a host kernel
- KVM and ESXi: NICs via PCI direct-map
- Vhost-user, netmap, virtio paravirtualized NICs
- Tun/tap drivers
- DPDK poll-mode device drivers

Integrated with the DPDK, VPP supports existing NIC devices including:

- Intel i40e, Intel ixgbe physical and virtual functions, Intel e1000, virtio, vhost-user, Linux TAP
- HP rebranded Intel Niantic MAC/PHY
- Cisco VIC

Security issues considered:

- Extensive white-box testing by Cisco's security team
- Image segment base address randomization
- Shared-memory segment base address randomization
- Stack bounds checking
- Debug CLI "chroot"

The vector method of packet processing has been proven as the primary punt/inject path on major architectures.

1.1.10 Supported Architectures

Table 1: Supported Architectures

The VPP platform supports:
x86/64

1.1.11 Supported Packaging Models

The VPP platform supports package installation on the following operating systems:

Table 2: Supported Packaging Models

Operating System:
Debian
Ubuntu 16.04
CentOS 7.3

1.1.12 Performance Expectations

One of the benefits of this implementation of VPP is its high performance on relatively low-power computing. This high level of performance is based on the following highlights:

- High-performance user-space network stack for commodity hardware
- The same code for host, VMs, Linux containers
- Integrated vhost-user virtio backend for high speed VM-to-VM connectivity
- L2 and L3 functionality, multiple encapsulations
- Leverages best-of-breed open source driver technology: DPDK
- Extensible by use of plugins
- Control-plane / orchestration-plane via standards-based APIs

1.1.13 Performance Metrics

The VPP platform has been shown to provide the following approximate performance metrics:

- Multiple MPPS from a single x86_64 core

- >100Gbps full-duplex on a single physical host
- Example of multi-core scaling benchmarks (on UCS-C240 M3, 3.5 gHz, all memory channels forwarded, simple ipv4 forwarding):
  - 1 core: 9 MPPS in+out
  - 2 cores: 13.4 MPPS in+out
  - 4 cores: 20.0 MPPS in+out

1.1.14 NDR Rates

NDR rates for 2p10GE, 1 core, L2 NIC-to_NIC

The following chart shows the NDR rates on: 2p10GE, 1 core, L2 NIC-to_NIC.

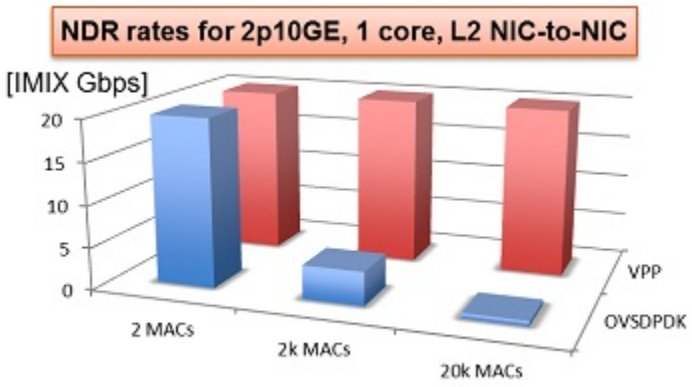

Fig. 7: NDR rate for 2p10GE, 1 core, L2 NIC-to_NIC

NDR rates for 2p10GE, 1 core, L2 NIC-to-VM/VM-to-VM

The following chart shows the NDR rates on: 2p10GE, 1 core, L2 NIC-to-VM/VM-to-VM .

NOTE:

- Virtual network infra benchmark of efficiency
- All tests per connection only, single core
- Potential higher performance with more connections, more cores
- Latest SW: OVSDPDK 2.4.0, VPP 09/2015

NDR rates VPP versus OVSDPDK

The following chart show VPP performance compared to open-source and commercial reports.

The rates reflect VPP and OVSDPDK performance tested on Haswell x86 platform with E5-2698v3 2x16C 2.3GHz. The graphs shows NDR rates for 12 port 10GE, 16 core, IPv4.

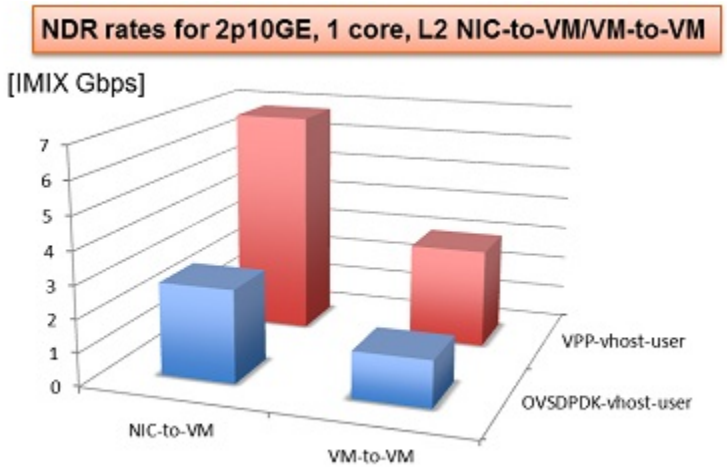

Fig. 8: NDR rates for 2p10GE, 1 core, L2 NIC-to-VM/VM-to-VM

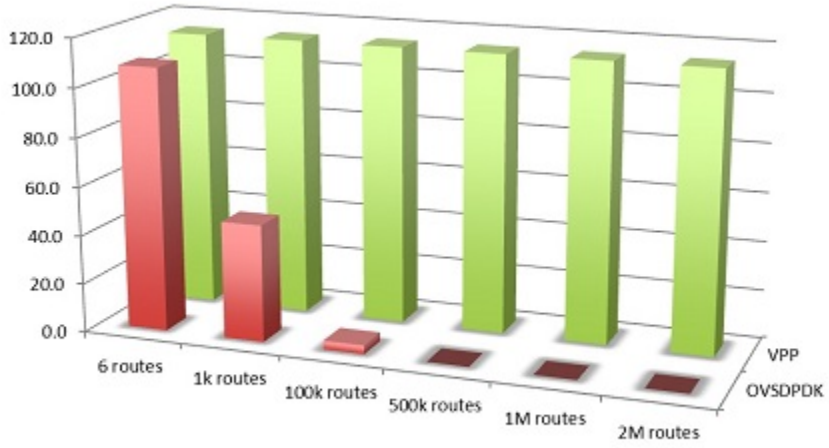

Fig. 9: VPP and OVSDPDK Comparison

CHAPTER 2

Setup

2.1 VPP Configuration Utility

2.1.1 Summary / Purpose of VPP Configuration Utility

The purpose of the VPP configuration utility is to allow the user to configure VPP in a simple and safe manner. The utility takes input from the user and then modifies the key configuration files. The user can then examine these files to be sure they are correct and then actually apply the configuration. The utility also includes an installation utility and some basic tests. This utility is currently released with release 17.10.

Table 1: Verified and Supported Operating Systems

Ubuntu16.04
centOS7
RHEL7

2.1.2 Use

The installation and executing of the VPP configuration utility is simple. First install the python pip module. Then using pip,

Ubuntu: Install and Run

Run the terminal as root

```
$ sudo -H bash
```

Afterwards, install vpp-config on root.

```
# pip install vpp-config
```

Once vpp-config is installed simply type:

```
# vpp-config

Welcome to the VPP system configuration utility

These are the files we will modify:
    /etc/vpp/startup.conf
    /etc/sysctl.d/80-vpp.conf
    /etc/default/grub

Before we change them, we'll create working copies in /usr/local/vpp/vpp-config/dryrun
Please inspect them carefully before applying the actual configuration (option 3)!

What would you like to do?

1) Show basic system information
2) Dry Run (Will save the configuration files in /usr/local/vpp/vpp-config/dryrun for␣
↪inspection)
        and user input in /usr/local/vpp/vpp-config/configs/auto-config.yaml
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
5) Execute some basic tests.
9 or q) Quit

Command:
and answer the questions. If you are not sure what to answer choose the
default.
```

Note: For yes or no questions the capital letter designates the default. For example, for a question that shows [Y/n] Y is the default. For numbers the default is within the brackets for example for a question that shows [1024]. 1024 is the default.

2.1.3 For Developers

Modifying the code is reasonable simple. Edit and debug the code from the root directory. In order to do this, we need a script that will copy or data files to the proper place. This is where they end up with pip install.

- Ubuntu

```
/usr/local/vpp/vpp-config
```

- Centos

```
/usr/vpp/vpp-config
```

Run this script to copy the relevant files correctly:

```
./scripts/cp-data.sh
```

Run this script to clean the environment.

```
./scripts/clean.sh
```

Note: This allows the developer to start from scratch.

Steps to Run the Utility

These are the steps to run the utility in this environment. The scripts are meant to be run from the *root directory*.

```
./scripts/clean.sh
./scripts/cp-data.sh
./vpp_config.py
```

When the utility is installed with pip the wrapper scripts/vpp-config is written to /usr/local/bin. However, the starting point when debugging this script locally is

```
   ./vpp_config.py

Run the utility by executing (from the root directory)

::

   ./vpp_config.py
```

The start point in the code is in vpp_config.py. Most of the work is done in the files in ./vpplib

Uploading to PyPi

To upload this utility to PyPi, simply do the following:

Note: Currently, I have my own account. When we want everyone to contribute we will need to change that.

```
$ sudo -H bash
# cd vpp_config
# python setup.py sdist bdist_wheel
# twine upload dist/*
```

2.1.4 Configuration Tool Main Menu

1. *Show basic system information*
2. *Dry Run*
3. *Full Configuration*
4. *List/Install/Uninstall VPP*
5. *Execute some basic tests*

2.1.5 Command 1. Show System Information

Before Configuration

When the utility is first started we can show the basic system information.

```
# vpp-config

Welcome to the VPP system configuration utility

These are the files we will modify:
    /etc/vpp/startup.conf
    /etc/sysctl.d/80-vpp.conf
    /etc/default/grub

Before we change them, we'll create working copies in /usr/local/vpp/vpp-config/dryrun
Please inspect them carefully before applying the actual configuration (option 3)!
```

```
What would you like to do?

1) Show basic system information
2) Dry Run (Will save the configuration files in /usr/local/vpp/vpp-config/dryrun for␣
↪inspection)
       and user input in /usr/local/vpp/vpp-config/configs/auto-config.yaml
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
5) Execute some basic tests.
9 or q) Quit

Command: 1

==============================
NODE: DUT1

CPU:
          Model name:    Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz
              CPU(s):    32
  Thread(s) per core:    2
  Core(s) per socket:    8
           Socket(s):    2
   NUMA node0 CPU(s):    0-7,16-23
   NUMA node1 CPU(s):    8-15,24-31
         CPU max MHz:    3600.0000
         CPU min MHz:    1200.0000
                 SMT:    Enabled

VPP Threads: (Name: Cpu Number)

Grub Command Line:
  Current: BOOT_IMAGE=/boot/vmlinuz-4.4.0-97-generic root=UUID=d760b82f-f37b-47e2-
↪9815-db8d479a3557 ro
  Configured: GRUB_CMDLINE_LINUX_DEFAULT=""

Huge Pages:
  Total System Memory           : 65863484 kB
  Total Free Memory             : 56862700 kB
  Actual Huge Page Total        : 1024
  Configured Huge Page Total    : 1024
  Huge Pages Free               : 1024
  Huge Page Size                : 2048 kB

Devices:
```

(continues on next page)

(continued from previous page)

```
Devices with link up (can not be used with VPP):
0000:08:00.0    enp8s0f0                  I350 Gigabit Network Connection

Devices bound to kernel drivers:
0000:90:00.0    enp144s0                  VIC Ethernet NIC
0000:8f:00.0    enp143s0                  VIC Ethernet NIC
0000:84:00.0    enp132s0f0,enp132s0f0d1   Ethernet Controller XL710 for 40GbE QSFP+
0000:84:00.1    enp132s0f1,enp132s0f1d1   Ethernet Controller XL710 for 40GbE QSFP+
0000:08:00.1    enp8s0f1                  I350 Gigabit Network Connection
0000:02:00.0    enp2s0f0                  82599ES 10-Gigabit SFI/SFP+ Network␣
↪Connection
0000:02:00.1    enp2s0f1                  82599ES 10-Gigabit SFI/SFP+ Network␣
↪Connection

No devices bound to DPDK drivers

VPP Service Status:
  Not Installed

==============================
```

After Configuration

When we show the system information after the system is configured notice that the VPP workers and the VPP main core is on the correct Numa Node. Notice also that VPP is running and the interfaces are shown.

```
What would you like to do?

1) Show basic system information
2) Dry Run (Will save the configuration files in /usr/local/vpp/vpp-config/dryrun for␣
↪inspection)
       and user input in /usr/local/vpp/vpp-config/configs/auto-config.yaml
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
5) Execute some basic tests.
9 or q) Quit

Command: 1

==============================
NODE: DUT1

CPU:
          Model name:    Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz
              CPU(s):    32
  Thread(s) per core:    2
  Core(s) per socket:    8
           Socket(s):    2
   NUMA node0 CPU(s):    0-7,16-23
   NUMA node1 CPU(s):    8-15,24-31
         CPU max MHz:    3600.0000
         CPU min MHz:    1200.0000
                 SMT:    Enabled

VPP Threads: (Name: Cpu Number)
```

(continues on next page)

(continued from previous page)

```
  vpp_main   : 8
  vpp_wk_1   : 10
  vpp_wk_0   : 9
  vpp_stats : 0

Grub Command Line:
  Current: BOOT_IMAGE=/boot/vmlinuz-4.4.0-97-generic root=UUID=d760b82f-f37b-47e2-
↪9815-db8d479a3557 ro
  Configured: GRUB_CMDLINE_LINUX_DEFAULT="isolcpus=8,9-10 nohz_full=8,9-10 rcu_
↪nocbs=8,9-10"

Huge Pages:
  Total System Memory           : 65863484 kB
  Total Free Memory             : 42048632 kB
  Actual Huge Page Total        : 8192
  Configured Huge Page Total    : 8192
  Huge Pages Free               : 7936
  Huge Page Size                : 2048 kB

Devices:
Total Number of Buffers: 25600

Devices with link up (can not be used with VPP):
0000:08:00.0    enp8s0f0                  I350 Gigabit Network Connection

Devices bound to kernel drivers:
0000:90:00.0    enp144s0                  VIC Ethernet NIC
0000:8f:00.0    enp143s0                  VIC Ethernet NIC
0000:84:00.0    enp132s0f0,enp132s0f0d1   Ethernet Controller XL710 for 40GbE QSFP+
0000:84:00.1    enp132s0f1,enp132s0f1d1   Ethernet Controller XL710 for 40GbE QSFP+
0000:08:00.1    enp8s0f1                  I350 Gigabit Network Connection
0000:02:00.0    enp2s0f0                  82599ES 10-Gigabit SFI/SFP+ Network␣
↪Connection
0000:02:00.1    enp2s0f1                  82599ES 10-Gigabit SFI/SFP+ Network␣
↪Connection

Devices bound to DPDK drivers:
0000:86:00.0                              82599ES 10-Gigabit SFI/SFP+ Network␣
↪Connection
0000:86:00.1                              82599ES 10-Gigabit SFI/SFP+ Network␣
↪Connection

Devices in use by VPP:
Name                         Socket RXQs RXDescs TXQs TXDescs
TenGigabitEthernet86/0/0          1    2    1024    3    1024
TenGigabitEthernet86/0/1          1    2    1024    3    1024

VPP Service Status:
  active (running)

==============================
```

2.1.6 Command 2. Dry Run

With VPP installed we can now execute a configuration dry run. This option will create the configuration files and put them in a dryrun directory. This directory is located for Ubuntu in /usr/local/vpp/vpp-config/dryrun and for Centos in

/usr/vpp/vpp-config/dryrun. These files should be examined to be sure that they are valid before actually applying the configuration with option 3.

```
What would you like to do?

1) Show basic system information
2) Dry Run (Will save the configuration files in /usr/local/vpp/vpp-config/dryrun for␣
↪inspection)
       and user input in /usr/local/vpp/vpp-config/configs/auto-config.yaml
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
5) Execute some basic tests.
9 or q) Quit

Command: 2

These device(s) are currently NOT being used by VPP or the OS.

PCI ID          Description
-----------------------------------------------------------
0000:86:00.0    82599ES 10-Gigabit SFI/SFP+ Network Connection
0000:86:00.1    82599ES 10-Gigabit SFI/SFP+ Network Connection

Would you like to give any of these devices back to the OS [Y/n]? y
Would you like to use device 0000:86:00.0 for the OS [y/N]? y
Would you like to use device 0000:86:00.1 for the OS [y/N]? y

These devices have kernel interfaces, but appear to be safe to use with VPP.

PCI ID          Kernel Interface(s)       Description
-------------------------------------------------------------------------------------
↪----
0000:90:00.0    enp144s0                  VIC Ethernet NIC
0000:8f:00.0    enp143s0                  VIC Ethernet NIC
0000:84:00.0    enp132s0f0,enp132s0f0d1   Ethernet Controller XL710 for 40GbE QSFP+
0000:84:00.1    enp132s0f1,enp132s0f1d1   Ethernet Controller XL710 for 40GbE QSFP+
0000:08:00.1    enp8s0f1                  I350 Gigabit Network Connection
0000:02:00.0    enp2s0f0                  82599ES 10-Gigabit SFI/SFP+ Network␣
↪Connection
0000:02:00.1    enp2s0f1                  82599ES 10-Gigabit SFI/SFP+ Network␣
↪Connection
0000:86:00.0    enp134s0f0                82599ES 10-Gigabit SFI/SFP+ Network␣
↪Connection
0000:86:00.1    enp134s0f1                82599ES 10-Gigabit SFI/SFP+ Network␣
↪Connection

Would you like to use any of these device(s) for VPP [y/N]? y
Would you like to use device 0000:90:00.0 for VPP [y/N]?
Would you like to use device 0000:8f:00.0 for VPP [y/N]?
Would you like to use device 0000:84:00.0 for VPP [y/N]?
Would you like to use device 0000:84:00.1 for VPP [y/N]?
Would you like to use device 0000:08:00.1 for VPP [y/N]?
Would you like to use device 0000:02:00.0 for VPP [y/N]?
Would you like to use device 0000:02:00.1 for VPP [y/N]?
Would you like to use device 0000:86:00.0 for VPP [y/N]? y
Would you like to use device 0000:86:00.1 for VPP [y/N]? y

These device(s) will be used by VPP.
```

(continues on next page)

(continued from previous page)

```
PCI ID          Description
-------------------------------------------------------------
0000:86:00.0    82599ES 10-Gigabit SFI/SFP+ Network Connection
0000:86:00.1    82599ES 10-Gigabit SFI/SFP+ Network Connection

Would you like to remove any of these device(s) [y/N]?

These device(s) will be used by VPP, please rerun this option if this is incorrect.

PCI ID          Description
-------------------------------------------------------------
0000:86:00.0    82599ES 10-Gigabit SFI/SFP+ Network Connection
0000:86:00.1    82599ES 10-Gigabit SFI/SFP+ Network Connection

Your system has 32 core(s) and 2 Numa Nodes.
To begin, we suggest not reserving any cores for VPP or other processes.
Then to improve performance try reserving cores as needed.

How many core(s) do you want to reserve for processes other than VPP? [0-16][0]?
How many core(s) shall we reserve for VPP workers[0-4][0]? 2
Should we reserve 1 core for the VPP Main thread? [y/N]? y

How many active-open / tcp client sessions are expected [0-10000000][0]?
How many passive-open / tcp server sessions are expected [0-10000000][0]?

There currently 1024 2048 kB huge pages free.
Do you want to reconfigure the number of huge pages [y/N]? y

There currently a total of 1024 huge pages.
How many huge pages do you want [1024 - 19414][1024]? 8192
```

2.1.7 Command 3. Apply Full Configuration

After the configuration files have been examined we can apply the configuration with option 3. Notice the default is NOT to change the grub command line. If the option to change the grub command line is selected a reboot will be required.

```
What would you like to do?

1) Show basic system information
2) Dry Run (Will save the configuration files in /usr/local/vpp/vpp-config/dryrun for␣
↪inspection)
       and user input in /usr/local/vpp/vpp-config/configs/auto-config.yaml
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
5) Execute some basic tests.
9 or q) Quit

Command: 3

We are now going to configure your system(s).

Are you sure you want to do this [Y/n]? y
These are the changes we will apply to
```

(continues on next page)

(continued from previous page)

```
the huge page file (/etc/sysctl.d/80-vpp.conf).

1,2d0
< vm.nr_hugepages=1024
4,7c2,3
< vm.max_map_count=3096
---
> vm.nr_hugepages=8192
> vm.max_map_count=17408
8a5
> kernel.shmmax=17179869184
10,15d6
< kernel.shmmax=2147483648


Are you sure you want to apply these changes [Y/n]?
These are the changes we will apply to
the VPP startup file (/etc/vpp/startup.conf).

---
>
>   main-core 8
>   corelist-workers 9-10
>
>   scheduler-policy fifo
>   scheduler-priority 50
>
67,68c56,66
< # dpdk {
---
> dpdk {
>
>   dev 0000:86:00.0 {
>     num-rx-queues 2
>   }
>   dev 0000:86:00.1 {
>     num-rx-queues 2
>   }
>   num-mbufs 25600
>
124c122
< # }
---
> }


Are you sure you want to apply these changes [Y/n]?

The configured grub cmdline looks like this:
GRUB_CMDLINE_LINUX_DEFAULT="isolcpus=8,9-10 nohz_full=8,9-10 rcu_nocbs=8,9-10"

The current boot cmdline looks like this:
BOOT_IMAGE=/boot/vmlinuz-4.4.0-97-generic root=UUID=d760b82f-f37b-47e2-9815-
↪db8d479a3557 ro

Do you want to keep the current boot cmdline [Y/n]?
```

2.1.8 Command 4. List/Install/Uninstall VPP

Notice when the basic system information was shown, VPP was not installed.

```
VPP Service Status:
  Not Installed

==============================
```

We can now install VPP with option 4

```
What would you like to do?

1) Show basic system information
2) Dry Run (Will save the configuration files in /usr/local/vpp/vpp-config/dryrun for␣
↪inspection)
       and user input in /usr/local/vpp/vpp-config/configs/auto-config.yaml
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
5) Execute some basic tests.
9 or q) Quit

Command: 4

There are no VPP packages on node localhost.
Do you want to install VPP [Y/n]? y
INFO:root: Local Command: ls /etc/apt/sources.list.d/99fd.io.list.orig
INFO:root:  /etc/apt/sources.list.d/99fd.io.list.orig
INFO:root: Local Command: rm /etc/apt/sources.list.d/99fd.io.list
INFO:root: Local Command: echo "deb [trusted=yes] https://nexus.fd.io/content/
↪repositories/fd.io.ubuntu.xenial.main/ ./
" | sudo tee /etc/apt/sources.list.d/99fd.io.list
INFO:root:  deb [trusted=yes] https://nexus.fd.io/content/repositories/fd.io.ubuntu.
↪xenial.main/ ./
.......
```

2.1.9 Command 5. Execute Basic tests

Command 1. Set IPv4 Addresses

Once VPP is configured we can add some ip addresses to the configured interfaces. Once this is done you should be able to ping the configured addresses and VPP is ready to use. After this option, is run a script is created in /usr/local/vpp/vpp-config/scripts/set_int_ipv4_and_up for Ubuntu and /usr/vpp/vpp-config/scripts/set_int_ipv4_and_up for Centos. This script can be used to configure the ip addresses in the future.

```
What would you like to do?

1) Show basic system information
2) Dry Run (Will save the configuration files in /usr/local/vpp/vpp-config/dryrun for␣
↪inspection)
       and user input in /usr/local/vpp/vpp-config/configs/auto-config.yaml
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
5) Execute some basic tests.
9 or q) Quit
```

(continues on next page)

(continued from previous page)

```
Command: 5

What would you like to do?

1) List/Create Simple IPv4 Setup
9 or q) Back to main menu.

Command: 1

These are the current interfaces with IP addresses:
TenGigabitEthernet86/0/0       Not Set              dn
TenGigabitEthernet86/0/1       Not Set              dn

Would you like to keep this configuration [Y/n]? n
Would you like add address to interface TenGigabitEthernet86/0/0 [Y/n]?
Please enter the IPv4 Address [n.n.n.n/n]: 30.0.0.2/24
Would you like add address to interface TenGigabitEthernet86/0/1 [Y/n]? y
Please enter the IPv4 Address [n.n.n.n/n]: 40.0.0.2/24

A script as been created at /usr/local/vpp/vpp-config/scripts/set_int_ipv4_and_up
This script can be run using the following:
vppctl exec /usr/local/vpp/vpp-config/scripts/set_int_ipv4_and_up


What would you like to do?

1) List/Create Simple IPv4 Setup
9 or q) Back to main menu.

Command: 1

These are the current interfaces with IP addresses:
TenGigabitEthernet86/0/0       30.0.0.2/24          up
TenGigabitEthernet86/0/1       40.0.0.2/24          up

Would you like to keep this configuration [Y/n]?
```

CHAPTER 3

Tasks

3.1 Writing and pushing VPP Documentation

3.1.1 Getting and Building the VPP Documentation

Overview

This repository contains the sources for much of the VPP documentation. These instructions show how most of the VPP documentation sources are obtained anbd built.

Build and View Instructions

I build and load the documents using a mac, but these instuctions should be portable to any platform. I used the Python virtual environment.

For more information on how to use the Python virtual enviroment check out Installing packages using pip and virtualenv.

1. Get the repository

```
git clone https://github.com/fdioDocs/vpp-docs
cd vpp-docs
```

2. Install the virtual environment

```
python -m pip install --user virtualenv
python -m virtualenv env
source env/bin/activate
pip install -r etc/requirements.txt
```

Note: To exit from the virtual environment execute:

```
deactivate
```

3. Build the html files

```
cd docs
make html
```

4. View the results.

To view the results start a browser and open the file:

```
<THE CLONED DIRECTORY>/docs/_build/html/index.html
```

3.1.2 Pushing a patch to the VPP Documentation

Pushing a Patch

I build and load the documents using a mac, but these instuctions should be portable to any platform. I used the Python virtual environment.

1. Review the changes

```
git status
```

2. Specify which files that will be pushed

```
git add <filename>
```

3. Commit the changes locally

```
git commit -s
```

4. Submit the changes for review

```
git review
```

Reviewing a Patch

1. Getting the patch for review

```
git review -d <review number>
```

1. Look at the changes

```
git status
```

2. Edit the changes you would like to add
3. Specify which files you changed

```
git add <filename>
```

4. Commit the changes locally

```
git commit --amend -s
```

5. Submit the changes for review

```
git review
```

Getting the Latest Sources

```
git reset --hard origin/master
git checkout master
```

3.2 Installing VPP Binaries from Packages

If you are simply using vpp, it can be convenient to simply install the packages. The instructions below will describe how pull, install and install the VPP packages.

3.2.1 Package Descriptions

The following is a brief description of the packages to be installed with VPP.

vpp

Vector Packet Processing–executables

- vpp - the vector packet engine
- vpp_api_test - vector packet engine API test tool
- vpp_json_test - vector packet engine JSON test tool

vpp-lib

Vector Packet Processing–runtime libraries. This package contains the VPP shared libraries, including:

- vppinfra - foundation library supporting vectors, hashes, bitmaps, pools, and string formatting.
- svm - vm library
- vlib - vector processing library
- vlib-api - binary API library
- vnet - network stack library

vpp-plugins

Vector Packet Processing–plugin modules

- acl
- dpdk
- flowprobe

- gtpu
- ixge
- kubeproxy
- l2e
- lb
- memif
- nat
- pppoe
- sixrd
- stn

vpp-dbg

Vector Packet Processing–debug symbols

vpp-dev

Vector Packet Processing–development support. This package contains development support files for the VPP libraries

vpp-api-java

JAVA binding for the VPP Binary API.

vpp-api-python

Python binding for the VPP Binary API.

vpp-api-lua

Lua binding for the VPP Binary API.

3.2.2 Installing on Ubuntu

The following are instructions on how to install VPP on Ubuntu.

The fd.io repo

1. Pick the Ubuntu version

- Ubuntu 16.04 - Xenial

```
export UBUNTU="xenial"
```

2. Pick the VPP version

- Latest VPP Release

```
unset -v RELEASE
```

- Latest VPP 18.04 Throttle Branch

```
export RELEASE=".stable.1804"
```

- Latest VPP 18.01 Throttle Branch

```
export RELEASE=".stable.1801"
```

- Latest VPP 17.10 Throttle Branch

```
export RELEASE=".stable.1710"
```

- Latest VPP 17.07 Throttle Branch

```
export RELEASE=".stable.1707"
```

- MASTER (in development)

```
export RELEASE=".master"
```

3. To write the fd.io sources list execute:

```
sudo rm /etc/apt/sources.list.d/99fd.io.list
echo "deb [trusted=yes] https://nexus.fd.io/content/repositories/fd.io$RELEASE.ubuntu.
↪$UBUNTU.main/ ./" | sudo tee -a /etc/apt/sources.list.d/99fd.io.list
```

Install the mandatory packages

```
sudo apt-get update
sudo apt-get install vpp vpp-lib vpp-plugin
```

Install the optional packages

```
sudo apt-get install vpp-dbg vpp-dev vpp-api-java vpp-api-python vpp-api-lua
```

Uninstall the packages

```
sudo apt-get remove --purge vpp*
```

3.2.3 Installing on Centos

The following are instructions on how to install VPP on Centos.

The fd.io repo

From the following choose one of the releases to install.

CentOS 7.3 - VPP Release RPMs (Latest)

Create a file **/etc/yum.repos.d/fdio-release.repo** with contents:

```
[fdio-release]
name=fd.io release branch latest merge
baseurl=https://nexus.fd.io/content/repositories/fd.io.centos7/
enabled=1
gpgcheck=0
```

CentOS 7.3 - VPP stable/1804 branch RPMs

Create a file **/etc/yum.repos.d/fdio-release.repo** with contents:

```
[fdio-stable-1804]
name=fd.io stable/1804 branch latest merge
baseurl=https://nexus.fd.io/content/repositories/fd.io.stable.1804.centos7/
enabled=1
gpgcheck=0
```

CentOS 7.3 - VPP master branch RPMs (in development)

Create a file **/etc/yum.repos.d/fdio-release.repo** with contents:

```
[fdio-master]
name=fd.io master branch latest merge
baseurl=https://nexus.fd.io/content/repositories/fd.io.master.centos7/
enabled=1
gpgcheck=0
```

Install VPP RPMs

```
sudo yum install vpp
```

Install the optional RPMs

```
sudo yum install vpp-plugins vpp-devel vpp-api-python vpp-api-lua vpp-api-java
```

Uninstall the VPP RPMs

```
sudo yum autoremove vpp*
```

3.2.4 Installing on openSUSE

The following are instructions on how to install VPP on openSUSE.

Blog post

For more information on VPP with openSUSE, please look at the following post.

- https://www.suse.com/communities/blog/vector-packet-processing-vpp-opensuse/

Installing

Top install VPP on openSUSE first pick the following release and execute the appropriate commands.

openSUSE Tumbleweed (rolling release)

```
sudo zypper install vpp vpp-plugins
```

openSUSE Leap 42.3

```
sudo zypper addrepo --name network https://download.opensuse.org/repositories/network/
↪openSUSE_Leap_42.3/network.repo
sudo zypper install vpp vpp-plugins
```

Uninstall

```
sudo zypper remove -u vpp vpp-plugins
```

openSUSE Tumbleweed (rolling release)

```
sudo zypper remove -u vpp vpp-plugins
```

openSUSE Leap 42.3

```
sudo zypper remove -u vpp vpp-plugins
sudo zypper removerepo network
```

3.2.5 Downloading the jvpp jar

The following are instructions on how to download the jvpp jar

Getting jvpp jar

VPP provides java bindings which can be downloaded at:

- https://nexus.fd.io/content/repositories/fd.io.release/io/fd/vpp/jvpp-core/18.01/jvpp-core-18.01.jar

Getting jvpp via maven

1. Add the following to the repositories section in your ~/.m2/settings.xml to pick up the fd.io maven repo:

```
 <repository>
  <id>fd.io-release</id>
  <name>fd.io-release</name>
  <url>https://nexus.fd.io/content/repositories/fd.io.release/</url>
  <releases>
    <enabled>false</enabled>
  </releases>
  <snapshots>
    <enabled>true</enabled>
  </snapshots>
</repository>
```

For more information on setting up maven repositories in settings.xml, please look at:

- https://maven.apache.org/guides/mini/guide-multiple-repositories.html

2. Then you can get jvpp by putting in the dependencies section of your pom.xml file:

```
<dependency>
  <groupId>io.fd.vpp</groupId>
  <artifactId>jvpp-core</artifactId>
  <version>17.10</version>
</dependency>
```

For more information on maven dependency managment, please look at:

- https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html

CHAPTER 4

Guides

4.1 Progressive VPP Tutorial

4.1.1 Introduction

This tutorial is designed for you to be able to run it on a single Ubuntu 16.04 VM on your laptop. It walks you through some very basic vpp senarios, with a focus on learning vpp commands, doing common actions, and being able to discover common things about the state of a running vpp.

This is *not* intended to be a 'how to run in a production environment' set of instructions.

4.1.2 Exercise: Setting up your environment

All of these exercises are designed to be performed on an Ubuntu 16.04 (Xenial) box.

If you have an Ubuntu 16.04 box on which you have sudo, you can feel free to use that.

If you do not, a Vagrantfile is provided to setup a basic Ubuntu 16.04 box for you

4.1.3 Vagrant Set Up

Action: Install Virtualbox

If you do not already have virtualbox on your laptop (or if it is not up to date), please download and install it:

https://www.virtualbox.org/wiki/Downloads

Action: Install Vagrant

If you do not already have Vagrant on your laptop (or if it is not up to date), please download it:

https://www.vagrantup.com/downloads.html

Action: Create a Vagrant Directory

Create a directory on your laptop

```
mkdir fdio-tutorial
cd fdio-tutorial/
```

Create a Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure(2) do |config|

config.vm.box = "puppetlabs/ubuntu-16.04-64-nocm"
config.vm.box_check_update = false

vmcpu=(ENV['VPP_VAGRANT_VMCPU'] || 2)
vmram=(ENV['VPP_VAGRANT_VMRAM'] || 4096)

config.ssh.forward_agent = true

config.vm.provider "virtualbox" do |vb|
    vb.customize ["modifyvm", :id, "--ioapic", "on"]
    vb.memory = "#{vmram}"
    vb.cpus = "#{vmcpu}"
    #support for the SSE4.x instruction is required in some versions of VB.
    vb.customize ["setextradata", :id, "VBoxInternal/CPUM/SSE4.1", "1"]
    vb.customize ["setextradata", :id, "VBoxInternal/CPUM/SSE4.2", "1"]
end
end
```

Action: Vagrant Up

Bring up your Vagrant VM:

```
vagrant up
```

Action: ssh to Vagrant VM

```
vagrant ssh
```

4.1.4 Exercise: Install VPP

Skills to be Learned

- Learn how to install vpp binary packges using apt-get.

Follow the instructions at *Installing VPP Binaries* for installing xenial vpp packages from the release repo. Please note, certain aspects of this tutorial require vpp 17.10 or later.

4.1.5 Exercise: VPP basics

Skills to be Learned

By the end of the exercise you should be able to:

- Run a vpp instance in a mode which allows multiple vpp processes to run
- Issue vpp commands from the unix shell
- Run a vpp shell and issue it commands

4.1.6 VPP command learned in this exercise

- show ver

4.1.7 Action: Remove dpdk plugin

In this tutorial, we will be running multiple vpp instances. DPDK does not work well with multiple instances, and so to run multiple instances we will need to disable the dpdk-plugin by removing it:

```
sudo rm -rf /usr/lib/vpp_plugins/dpdk_plugin.so
```

..how-to-run-vpp:

4.1.8 Action: Run VPP

VPP runs in userspace. In a production environment you will often run it with DPDK to connect to real NICs or vhost to connect to VMs. In those circumstances you usually run a single instance of vpp.

For purposes of this tutorial, it is going to be extremely useful to run multiple instances of vpp, and connect them to each other to form a topology. Fortunately, vpp supports this.

When running multiple vpp instances, each instance needs to have specified a 'name' or 'prefix'. In the example below, the 'name' or 'prefix' is "vpp1". Note that only one instance can use the dpdk plugin, since this plugin is trying to acquire a lock on a file.

```
sudo vpp unix {cli-listen /run/vpp/cli-vpp1.sock} api-segment { prefix vpp1 }
```

Example Output:

```
vlib_plugin_early_init:230: plugin path /usr/lib/vpp_plugins
```

Please note:

- "api-segment {prefix vpp1}" tells vpp how to name the files in /dev/shm/ for your vpp instance differently from the default.
- "unix {cli-listen /run/vpp/cli-vpp1.sock}" tells vpp to use a non-default socket file when being addressed by vppctl.

If you can't see the vpp process running on the host, activate the nodaemon option to better understand what is happening

```
sudo vpp unix {nodaemon cli-listen /run/vpp/cli-vpp1.sock} api-segment { prefix vpp1 }
```

Example Output with errors from the dpdk plugin:

```
vlib_plugin_early_init:356: plugin path /usr/lib/vpp_plugins
load_one_plugin:184: Loaded plugin: acl_plugin.so (Access Control Lists)
load_one_plugin:184: Loaded plugin: dpdk_plugin.so (Data Plane Development Kit (DPDK))
load_one_plugin:184: Loaded plugin: flowprobe_plugin.so (Flow per Packet)
load_one_plugin:184: Loaded plugin: gtpu_plugin.so (GTPv1-U)
load_one_plugin:184: Loaded plugin: ila_plugin.so (Identifier-locator addressing for␣
↪IPv6)
load_one_plugin:184: Loaded plugin: ioam_plugin.so (Inbound OAM)
load_one_plugin:114: Plugin disabled (default): ixge_plugin.so
load_one_plugin:184: Loaded plugin: kubeproxy_plugin.so (kube-proxy data plane)
load_one_plugin:184: Loaded plugin: l2e_plugin.so (L2 Emulation)
load_one_plugin:184: Loaded plugin: lb_plugin.so (Load Balancer)
load_one_plugin:184: Loaded plugin: libsixrd_plugin.so (IPv6 Rapid Deployment on IPv4␣
↪Infrastructure (RFC5969))
load_one_plugin:184: Loaded plugin: memif_plugin.so (Packet Memory Interface␣
↪(experimetal))
load_one_plugin:184: Loaded plugin: nat_plugin.so (Network Address Translation)
load_one_plugin:184: Loaded plugin: pppoe_plugin.so (PPPoE)
load_one_plugin:184: Loaded plugin: stn_plugin.so (VPP Steals the NIC for Container␣
↪integration)
vpp[10211]: vlib_pci_bind_to_uio: Skipping PCI device 0000:00:03.0 as host interface␣
↪eth0 is up
vpp[10211]: vlib_pci_bind_to_uio: Skipping PCI device 0000:00:04.0 as host interface␣
↪eth1 is up
vpp[10211]: dpdk_config:1240: EAL init args: -c 1 -n 4 --huge-dir /run/vpp/hugepages -
↪-file-prefix vpp -b 0000:00:03.0 -b 0000:00:04.0 --master-lcore 0 --socket-mem 64
EAL: No free hugepages reported in hugepages-1048576kB
EAL: Error - exiting with code: 1
Cause: Cannot create lock on '/var/run/.vpp_config'. Is another primary process␣
↪running?
```

4.1.9 Action: Send commands to VPP using vppctl

You can send vpp commands with a utility called *vppctl*.

When running vppctl against a named version of vpp, you will need to run:

```
sudo vppctl -s /run/vpp/cli-${name}.sock ${cmd}
```

Note

```
/run/vpp/cli-${name}.sock
```

is the particular naming convention used in this tutorial. By default you can set vpp to use what ever socket file name you would like at startup (the default config file uses /run/vpp/cli.sock) if two different vpps are being run (as in this tutorial) you must use distinct socket files for each one.

So to run 'show ver' against the vpp instance named vpp1 you would run:

```
sudo vppctl -s /run/vpp/cli-vpp1.sock show ver
```

Output:

```
vpp v17.04-rc0~177-g006eb47 built by ubuntu on fdio-ubuntu1604-sevt at Mon Jan 30␣
↪18:30:12 UTC 2017
```

4.1.10 Action: Start a VPP shell using vppctl

You can also use vppctl to launch a vpp shell with which you can run multiple vpp commands interactively by running:

```
sudo vppctl -s /run/vpp/cli-${name}.sock
```

which will give you a command prompt.

Try doing show ver that way:

```
sudo vppctl -s /run/vpp/cli-vpp1.sock
vpp# show ver
```

Output:

```
vpp v17.04-rc0~177-g006eb47 built by ubuntu on fdio-ubuntu1604-sevt at Mon Jan 30␣
↪18:30:12 UTC 2017

vpp#
```

To exit the vppctl shell:

```
vpp# quit
```

4.1.11 Exercise: Create an interface

Skills to be Learned

1. Create a veth interface in Linux host
2. Assign an IP address to one end of the veth interface in the Linux host
3. Create a vpp host-interface that connected to one end of a veth interface via AF_PACKET
4. Add an ip address to a vpp interface
5. Setup a 'trace'
6. View a 'trace'
7. Clear a 'trace'
8. Verify using ping from host
9. Ping from vpp
10. Examine Arp Table
11. Examine ip fib

VPP command learned in this exercise

1. create host-interface
2. set int state
3. set int ip address
4. show hardware
5. show int

6. show int addr
7. trace add
8. clear trace
9. ping
10. show ip arp
11. show ip fib

Topology

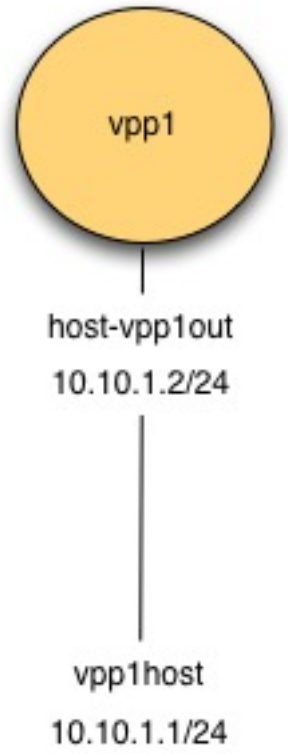

Fig. 1: Figure: Create Interface Topology

Initial State

The initial state here is presumed to be the final state from the exercise VPP Basics

Action: Create veth interfaces on host

In Linux, there is a type of interface call 'veth'. Think of a 'veth' interface as being an interface that has two ends to it (rather than one).

Create a veth interface with one end named **vpp1out** and the other named **vpp1host**

```
sudo ip link add name vpp1out type veth peer name vpp1host
```

Turn up both ends:

```
sudo ip link set dev vpp1out up
sudo ip link set dev vpp1host up
```

Assign an IP address

```
sudo ip addr add 10.10.1.1/24 dev vpp1host
```

Display the result:

```
sudo ip addr show vpp1host
```

Example Output:

```
10: vpp1host@vpp1out: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state␣
↪UP group default qlen 1000
    link/ether 5e:97:e3:41:aa:b8 brd ff:ff:ff:ff:ff:ff
    inet 10.10.1.1/24 scope global vpp1host
       valid_lft forever preferred_lft forever
    inet6 fe80::5c97:e3ff:fe41:aab8/64 scope link
       valid_lft forever preferred_lft forever
```

Action: Create vpp host- interface

Create a host interface attached to **vpp1out**.

```
sudo vppctl -s /run/vpp/cli-vpp1.sock create host-interface name vpp1out
```

Output:

```
host-vpp1out
```

Confirm the interface:

```
sudo vppctl -s /run/vpp/cli-vpp1.sock show hardware
```

Example Output:

```
              Name                Idx   Link  Hardware
host-vpp1out                       1     up   host-vpp1out
  Ethernet address 02:fe:48:ec:d5:a7
  Linux PACKET socket interface
local0                             0    down  local0
  local
```

Turn up the interface:

```
sudo vppctl -s /run/vpp/cli-vpp1.sock set int state host-vpp1out up
```

Confirm the interface is up:

```
sudo vppctl -s /run/vpp/cli-vpp1.sock show int
```

```
              Name               Idx       State          Counter          Count
host-vpp1out                      1         up
local0                            0         down
```

Assign ip address 10.10.1.2/24

```
sudo vppctl -s /run/vpp/cli-vpp1.sock set int ip address host-vpp1out 10.10.1.2/24
```

Confirm the ip address is assigned:

```
sudo vppctl -s /run/vpp/cli-vpp1.sock show int addr
```

```
host-vpp1out (up):
  10.10.1.2/24
local0 (dn):
```

Action: Add trace

```
sudo vppctl -s /run/vpp/cli-vpp1.sock trace add af-packet-input 10
```

Action: Ping from host to vpp

```
ping -c 1 10.10.1.2
```

```
PING 10.10.1.2 (10.10.1.2) 56(84) bytes of data.
64 bytes from 10.10.1.2: icmp_seq=1 ttl=64 time=0.557 ms

--- 10.10.1.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.557/0.557/0.557/0.000 ms
```

Action: Examine Trace of ping from host to vpp

```
sudo vppctl -s /run/vpp/cli-vpp1.sock show trace
```

```
------------------- Start of thread 0 vpp_main -------------------
Packet 1

00:09:30:397798: af-packet-input
  af_packet: hw_if_index 1 next-index 4
    tpacket2_hdr:
      status 0x20000001 len 42 snaplen 42 mac 66 net 80
      sec 0x588fd3ac nsec 0x375abde7 vlan 0 vlan_tpid 0
00:09:30:397906: ethernet-input
  ARP: fa:13:55:ac:d9:50 -> ff:ff:ff:ff:ff:ff
00:09:30:397912: arp-input
  request, type ethernet/IP4, address size 6/4
  fa:13:55:ac:d9:50/10.10.1.1 -> 00:00:00:00:00:00/10.10.1.2
00:09:30:398191: host-vpp1out-output
```

(continues on next page)

(continued from previous page)

```
  host-vpp1out
  ARP: 02:fe:48:ec:d5:a7 -> fa:13:55:ac:d9:50
  reply, type ethernet/IP4, address size 6/4
  02:fe:48:ec:d5:a7/10.10.1.2 -> fa:13:55:ac:d9:50/10.10.1.1

Packet 2

00:09:30:398227: af-packet-input
  af_packet: hw_if_index 1 next-index 4
    tpacket2_hdr:
      status 0x20000001 len 98 snaplen 98 mac 66 net 80
      sec 0x588fd3ac nsec 0x37615060 vlan 0 vlan_tpid 0
00:09:30:398295: ethernet-input
  IP4: fa:13:55:ac:d9:50 -> 02:fe:48:ec:d5:a7
00:09:30:398298: ip4-input
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x9b46
    fragment id 0x894c, flags DONT_FRAGMENT
  ICMP echo_request checksum 0x83c
00:09:30:398300: ip4-lookup
  fib 0 dpo-idx 5 flow hash: 0x00000000
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x9b46
    fragment id 0x894c, flags DONT_FRAGMENT
  ICMP echo_request checksum 0x83c
00:09:30:398303: ip4-local
    ICMP: 10.10.1.1 -> 10.10.1.2
      tos 0x00, ttl 64, length 84, checksum 0x9b46
      fragment id 0x894c, flags DONT_FRAGMENT
    ICMP echo_request checksum 0x83c
00:09:30:398305: ip4-icmp-input
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x9b46
    fragment id 0x894c, flags DONT_FRAGMENT
  ICMP echo_request checksum 0x83c
00:09:30:398307: ip4-icmp-echo-request
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x9b46
    fragment id 0x894c, flags DONT_FRAGMENT
  ICMP echo_request checksum 0x83c
00:09:30:398317: ip4-load-balance
  fib 0 dpo-idx 10 flow hash: 0x0000000e
  ICMP: 10.10.1.2 -> 10.10.1.1
    tos 0x00, ttl 64, length 84, checksum 0xbef3
    fragment id 0x659f, flags DONT_FRAGMENT
  ICMP echo_reply checksum 0x103c
00:09:30:398318: ip4-rewrite
  tx_sw_if_index 1 dpo-idx 2 : ipv4 via 10.10.1.1 host-vpp1out: IP4:␣
↪02:fe:48:ec:d5:a7 -> fa:13:55:ac:d9:50 flow hash: 0x00000000
  IP4: 02:fe:48:ec:d5:a7 -> fa:13:55:ac:d9:50
  ICMP: 10.10.1.2 -> 10.10.1.1
    tos 0x00, ttl 64, length 84, checksum 0xbef3
    fragment id 0x659f, flags DONT_FRAGMENT
  ICMP echo_reply checksum 0x103c
00:09:30:398320: host-vpp1out-output
  host-vpp1out
  IP4: 02:fe:48:ec:d5:a7 -> fa:13:55:ac:d9:50
```

(continues on next page)

(continued from previous page)

```
  ICMP: 10.10.1.2 -> 10.10.1.1
    tos 0x00, ttl 64, length 84, checksum 0xbef3
    fragment id 0x659f, flags DONT_FRAGMENT
  ICMP echo_reply checksum 0x103c
```

Action: Clear trace buffer

```
sudo vppctl -s /run/vpp/cli-vpp1.sock clear  trace
```

Action: ping from vpp to host

```
sudo vppctl -s /run/vpp/cli-vpp1.sock ping 10.10.1.1
```

```
64 bytes from 10.10.1.1: icmp_seq=1 ttl=64 time=.0865 ms
64 bytes from 10.10.1.1: icmp_seq=2 ttl=64 time=.0914 ms
64 bytes from 10.10.1.1: icmp_seq=3 ttl=64 time=.0943 ms
64 bytes from 10.10.1.1: icmp_seq=4 ttl=64 time=.0959 ms
64 bytes from 10.10.1.1: icmp_seq=5 ttl=64 time=.0858 ms

Statistics: 5 sent, 5 received, 0% packet loss
```

Action: Examine Trace of ping from vpp to host

```
sudo vppctl -s /run/vpp/cli-vpp1.sock show trace
```

```
------------------- Start of thread 0 vpp_main -------------------
Packet 1

00:12:47:155326: af-packet-input
  af_packet: hw_if_index 1 next-index 4
    tpacket2_hdr:
      status 0x20000001 len 98 snaplen 98 mac 66 net 80
      sec 0x588fd471 nsec 0x161c61ad vlan 0 vlan_tpid 0
00:12:47:155331: ethernet-input
  IP4: fa:13:55:ac:d9:50 -> 02:fe:48:ec:d5:a7
00:12:47:155334: ip4-input
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x2604
    fragment id 0x3e8f
  ICMP echo_reply checksum 0x1a83
00:12:47:155335: ip4-lookup
  fib 0 dpo-idx 5 flow hash: 0x00000000
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x2604
    fragment id 0x3e8f
  ICMP echo_reply checksum 0x1a83
00:12:47:155336: ip4-local
    ICMP: 10.10.1.1 -> 10.10.1.2
      tos 0x00, ttl 64, length 84, checksum 0x2604
```

(continues on next page)

(continued from previous page)

```
      fragment id 0x3e8f
    ICMP echo_reply checksum 0x1a83
00:12:47:155339: ip4-icmp-input
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x2604
    fragment id 0x3e8f
  ICMP echo_reply checksum 0x1a83
00:12:47:155342: ip4-icmp-echo-reply
  ICMP echo id 17572 seq 1
00:12:47:155349: error-drop
  ip4-icmp-input: unknown type

Packet 2

00:12:48:155330: af-packet-input
  af_packet: hw_if_index 1 next-index 4
    tpacket2_hdr:
      status 0x20000001 len 98 snaplen 98 mac 66 net 80
      sec 0x588fd472 nsec 0x1603e95b vlan 0 vlan_tpid 0
00:12:48:155337: ethernet-input
  IP4: fa:13:55:ac:d9:50 -> 02:fe:48:ec:d5:a7
00:12:48:155341: ip4-input
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x2565
    fragment id 0x3f2e
  ICMP echo_reply checksum 0x7405
00:12:48:155343: ip4-lookup
  fib 0 dpo-idx 5 flow hash: 0x00000000
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x2565
    fragment id 0x3f2e
  ICMP echo_reply checksum 0x7405
00:12:48:155344: ip4-local
    ICMP: 10.10.1.1 -> 10.10.1.2
      tos 0x00, ttl 64, length 84, checksum 0x2565
      fragment id 0x3f2e
    ICMP echo_reply checksum 0x7405
00:12:48:155346: ip4-icmp-input
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x2565
    fragment id 0x3f2e
  ICMP echo_reply checksum 0x7405
00:12:48:155348: ip4-icmp-echo-reply
  ICMP echo id 17572 seq 2
00:12:48:155351: error-drop
  ip4-icmp-input: unknown type

Packet 3

00:12:49:155331: af-packet-input
  af_packet: hw_if_index 1 next-index 4
    tpacket2_hdr:
      status 0x20000001 len 98 snaplen 98 mac 66 net 80
      sec 0x588fd473 nsec 0x15eb77ef vlan 0 vlan_tpid 0
00:12:49:155337: ethernet-input
  IP4: fa:13:55:ac:d9:50 -> 02:fe:48:ec:d5:a7
00:12:49:155341: ip4-input
```

(continues on next page)

(continued from previous page)

```
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x249e
    fragment id 0x3ff5
  ICMP echo_reply checksum 0xf446
00:12:49:155343: ip4-lookup
  fib 0 dpo-idx 5 flow hash: 0x00000000
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x249e
    fragment id 0x3ff5
  ICMP echo_reply checksum 0xf446
00:12:49:155345: ip4-local
    ICMP: 10.10.1.1 -> 10.10.1.2
      tos 0x00, ttl 64, length 84, checksum 0x249e
      fragment id 0x3ff5
    ICMP echo_reply checksum 0xf446
00:12:49:155349: ip4-icmp-input
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x249e
    fragment id 0x3ff5
  ICMP echo_reply checksum 0xf446
00:12:49:155350: ip4-icmp-echo-reply
  ICMP echo id 17572 seq 3
00:12:49:155354: error-drop
  ip4-icmp-input: unknown type

Packet 4

00:12:50:155335: af-packet-input
  af_packet: hw_if_index 1 next-index 4
    tpacket2_hdr:
      status 0x20000001 len 98 snaplen 98 mac 66 net 80
      sec 0x588fd474 nsec 0x15d2ffb6 vlan 0 vlan_tpid 0
00:12:50:155341: ethernet-input
  IP4: fa:13:55:ac:d9:50 -> 02:fe:48:ec:d5:a7
00:12:50:155346: ip4-input
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x2437
    fragment id 0x405c
  ICMP echo_reply checksum 0x5b6e
00:12:50:155347: ip4-lookup
  fib 0 dpo-idx 5 flow hash: 0x00000000
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x2437
    fragment id 0x405c
  ICMP echo_reply checksum 0x5b6e
00:12:50:155350: ip4-local
    ICMP: 10.10.1.1 -> 10.10.1.2
      tos 0x00, ttl 64, length 84, checksum 0x2437
      fragment id 0x405c
    ICMP echo_reply checksum 0x5b6e
00:12:50:155351: ip4-icmp-input
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x2437
    fragment id 0x405c
  ICMP echo_reply checksum 0x5b6e
00:12:50:155353: ip4-icmp-echo-reply
  ICMP echo id 17572 seq 4
```

(continues on next page)

(continued from previous page)

```
00:12:50:155356: error-drop
  ip4-icmp-input: unknown type

Packet 5

00:12:51:155324: af-packet-input
  af_packet: hw_if_index 1 next-index 4
    tpacket2_hdr:
      status 0x20000001 len 98 snaplen 98 mac 66 net 80
      sec 0x588fd475 nsec 0x15ba8726 vlan 0 vlan_tpid 0
00:12:51:155331: ethernet-input
  IP4: fa:13:55:ac:d9:50 -> 02:fe:48:ec:d5:a7
00:12:51:155335: ip4-input
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x23cc
    fragment id 0x40c7
  ICMP echo_reply checksum 0xedb3
00:12:51:155337: ip4-lookup
  fib 0 dpo-idx 5 flow hash: 0x00000000
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x23cc
    fragment id 0x40c7
  ICMP echo_reply checksum 0xedb3
00:12:51:155338: ip4-local
    ICMP: 10.10.1.1 -> 10.10.1.2
      tos 0x00, ttl 64, length 84, checksum 0x23cc
      fragment id 0x40c7
    ICMP echo_reply checksum 0xedb3
00:12:51:155341: ip4-icmp-input
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x23cc
    fragment id 0x40c7
  ICMP echo_reply checksum 0xedb3
00:12:51:155343: ip4-icmp-echo-reply
  ICMP echo id 17572 seq 5
00:12:51:155346: error-drop
  ip4-icmp-input: unknown type

Packet 6

00:12:52:175185: af-packet-input
  af_packet: hw_if_index 1 next-index 4
    tpacket2_hdr:
      status 0x20000001 len 42 snaplen 42 mac 66 net 80
      sec 0x588fd476 nsec 0x16d05dd0 vlan 0 vlan_tpid 0
00:12:52:175195: ethernet-input
  ARP: fa:13:55:ac:d9:50 -> 02:fe:48:ec:d5:a7
00:12:52:175200: arp-input
  request, type ethernet/IP4, address size 6/4
  fa:13:55:ac:d9:50/10.10.1.1 -> 00:00:00:00:00:00/10.10.1.2
00:12:52:175214: host-vpp1out-output
  host-vpp1out
  ARP: 02:fe:48:ec:d5:a7 -> fa:13:55:ac:d9:50
  reply, type ethernet/IP4, address size 6/4
  02:fe:48:ec:d5:a7/10.10.1.2 -> fa:13:55:ac:d9:50/10.10.1.1
```

After examinging the trace, clear it again.

Action: Examine arp tables

```
sudo vppctl -s /run/vpp/cli-vpp1.sock show ip arp
```

```
Time           IP4       Flags      Ethernet              Interface
570.4092    10.10.1.1      D    fa:13:55:ac:d9:50       host-vpp1out
```

Action: Examine routing table

```
sudo vppctl -s /run/vpp/cli-vpp1.sock show ip fib
```

```
ipv4-VRF:0, fib_index 0, flow hash: src dst sport dport proto
0.0.0.0/0
  unicast-ip4-chain
  [@0]: dpo-load-balance: [index:0 buckets:1 uRPF:0 to:[0:0]]
    [0] [@0]: dpo-drop ip4
0.0.0.0/32
  unicast-ip4-chain
  [@0]: dpo-load-balance: [index:1 buckets:1 uRPF:1 to:[0:0]]
    [0] [@0]: dpo-drop ip4
10.10.1.1/32
  unicast-ip4-chain
  [@0]: dpo-load-balance: [index:10 buckets:1 uRPF:9 to:[5:420] via:[1:84]]
    [0] [@5]: ipv4 via 10.10.1.1 host-vpp1out: IP4: 02:fe:48:ec:d5:a7 ->␣
↪fa:13:55:ac:d9:50
10.10.1.0/24
  unicast-ip4-chain
  [@0]: dpo-load-balance: [index:8 buckets:1 uRPF:7 to:[0:0]]
    [0] [@4]: ipv4-glean: host-vpp1out
10.10.1.2/32
  unicast-ip4-chain
  [@0]: dpo-load-balance: [index:9 buckets:1 uRPF:8 to:[6:504]]
    [0] [@2]: dpo-receive: 10.10.1.2 on host-vpp1out
224.0.0.0/4
  unicast-ip4-chain
  [@0]: dpo-load-balance: [index:3 buckets:1 uRPF:3 to:[0:0]]
    [0] [@0]: dpo-drop ip4
240.0.0.0/4
  unicast-ip4-chain
  [@0]: dpo-load-balance: [index:2 buckets:1 uRPF:2 to:[0:0]]
    [0] [@0]: dpo-drop ip4
255.255.255.255/32
  unicast-ip4-chain
  [@0]: dpo-load-balance: [index:4 buckets:1 uRPF:4 to:[0:0]]
    [0] [@0]: dpo-drop ip4
```

4.1.12 Exercise: Connecting two vpp instances

Background

memif is a very high performance, direct memory interface type which can be used between vpp instances to form a topology. It uses a file socket for a control channel to set up that shared memory.

Skills to be Learned

You will learn the following new skill in this exercise:

1. Create a memif interface between two vpp instances

You should be able to perform this exercise with the following skills learned in previous exercises:

1. Run a second vpp instance
2. Add an ip address to a vpp interface
3. Ping from vpp

Topology

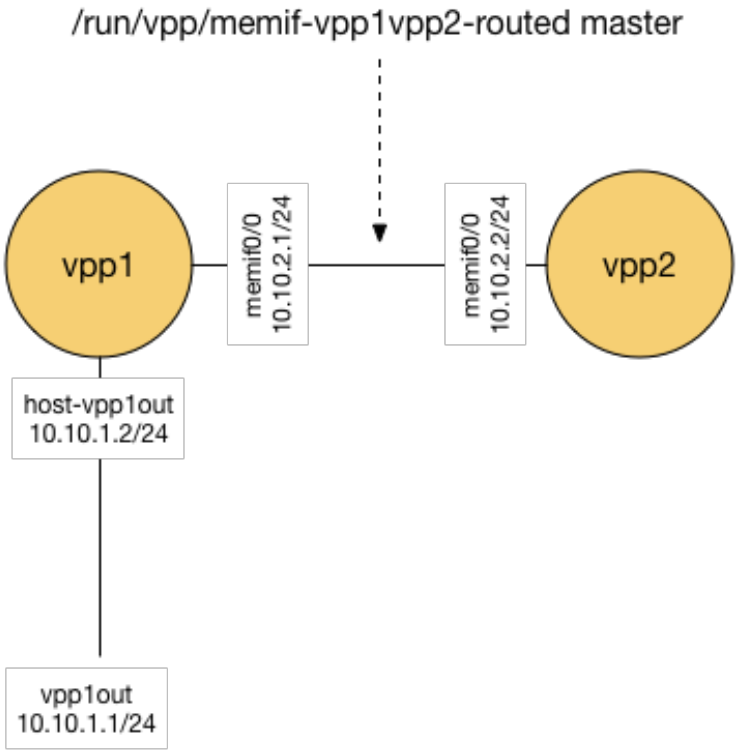

Fig. 2: Connect two vpp topolgy

Initial state

The initial state here is presumed to be the final state from the exercise Create an Interface

Action: Running a second vpp instances

You should already have a vpp instance running named: vpp1.

Run a second vpp instance named: vpp2.

Action: Create memif interface on vpp1

Create a memif interface on vpp1:

```
sudo vppctl -s /run/vpp/cli-vpp1.sock create memif id 0 master
```

This will create an interface on vpp1 memif0/0 using /run/vpp/memif as its socket file. The role of vpp1 for this memif inteface is 'master'.

Use your previously used skills to:

1. Set the memif0/0 state to up.
2. Assign IP address 10.10.2.1/24 to memif0/0
3. Examine memif0/0 via show commands

Action: Create memif interface on vpp2

We want vpp2 to pick up the 'slave' role using the same run/vpp/memif-vpp1vpp2 socket file

```
sudo vppctl -s /run/vpp/cli-vpp2.sock create memif id 0 slave
```

This will create an interface on vpp2 memif0/0 using /run/vpp/memif as its socket file. The role of vpp1 for this memif inteface is 'slave'.

Use your previously used skills to:

1. Set the memif0/0 state to up.
2. Assign IP address 10.10.2.2/24 to memif0/0
3. Examine memif0/0 via show commands

Action: Ping from vpp1 to vpp2

Ping 10.10.2.2 from vpp1

Ping 10.10.2.1 from vpp2

4.1.13 Exercise: Routing

Skills to be Learned

In this exercise you will learn these new skills:

1. Add route to Linux Host routing table
2. Add route to vpp routing table

And revisit the old ones:

1. Examine vpp routing table
2. Enable trace on vpp1 and vpp2
3. ping from host to vpp
4. Examine and clear trace on vpp1 and vpp2
5. ping from vpp to host
6. Examine and clear trace on vpp1 and vpp2

vpp command learned in this exercise

1. ip route add

Topology

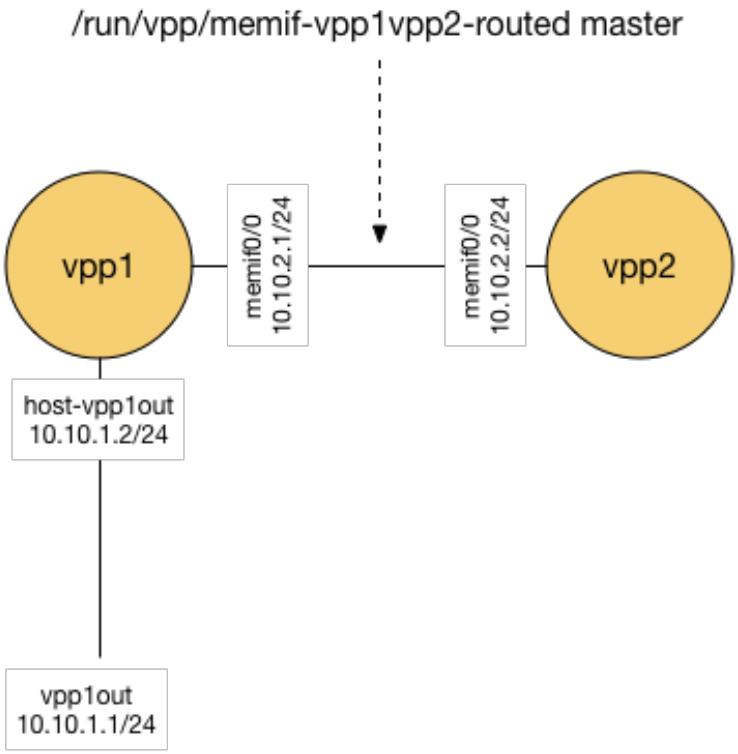

Fig. 3: Connect two vpp topology

Initial State

The initial state here is presumed to be the final state from the exercise Connecting two vpp instances

Action: Setup host route

```
sudo ip route add 10.10.2.0/24 via 10.10.1.2
ip route
```

```
default via 10.0.2.2 dev enp0s3
10.0.2.0/24 dev enp0s3  proto kernel  scope link  src 10.0.2.15
10.10.1.0/24 dev vpp1host  proto kernel  scope link  src 10.10.1.1
10.10.2.0/24 via 10.10.1.2 dev vpp1host
```

Setup return route on vpp2

```
sudo vppctl -s /run/vpp/cli-vpp2.sock ip route add 10.10.1.0/24  via 10.10.2.1
```

Ping from host through vpp1 to vpp2

1. Setup a trace on vpp1 and vpp2
2. Ping 10.10.2.2 from the host
3. Examine the trace on vpp1 and vpp2
4. Clear the trace on vpp1 and vpp2

4.1.14 Exercise: Switching

Skills to be Learned

1. Associate an interface with a bridge domain
2. Create a loopback interaface
3. Create a BVI (Bridge Virtual Interface) for a bridge domain
4. Examine a bridge domain

vpp command learned in this exercise

1. show bridge
2. show bridge detail
3. set int l2 bridge
4. show l2fib verbose

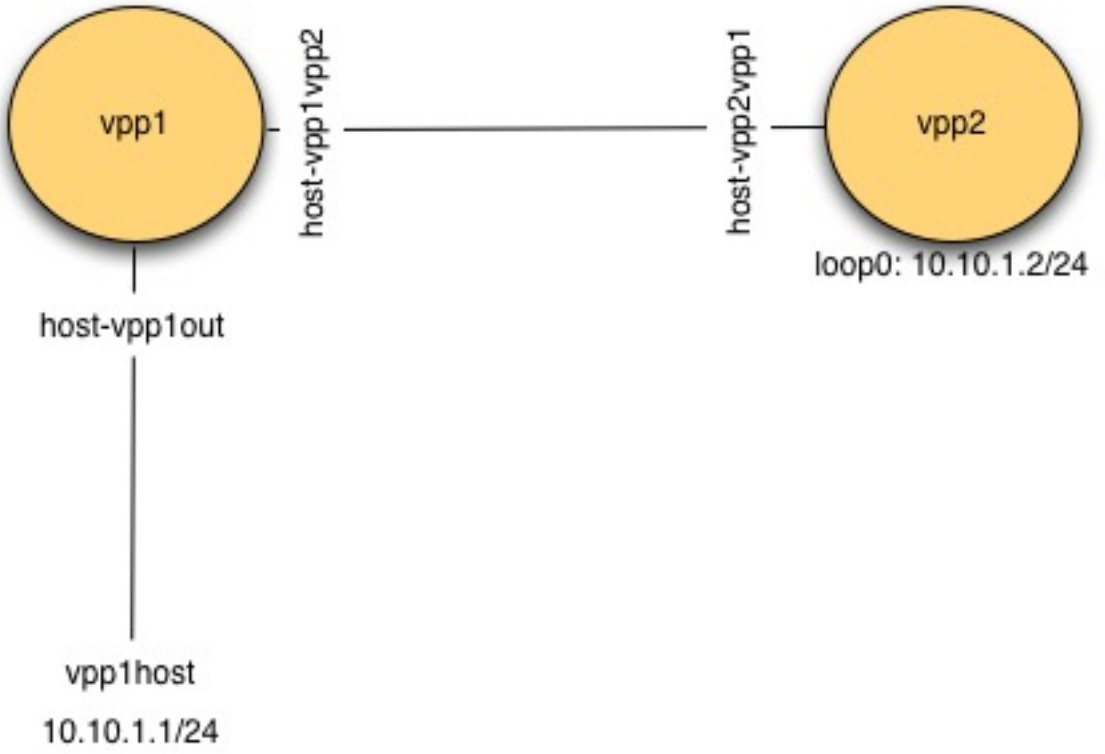

Fig. 4: Switching Topology

Topology

Initial state

Unlike previous exercises, for this one you want to start tabula rasa.

Note: You will lose all your existing config in your vpp instances!

To clear existing config from previous exercises run:

```
ps -ef | grep vpp | awk '{print $2}'| xargs sudo kill
sudo ip link del dev vpp1host
sudo ip link del dev vpp1vpp2
```

Action: Run vpp instances

1. Run a vpp instance named **vpp1**
2. Run a vpp instance named **vpp2**

Action: Connect vpp1 to host

1. Create a veth with one end named vpp1host and the other named vpp1out.
2. Connect vpp1out to vpp1
3. Add ip address 10.10.1.1/24 on vpp1host

Action: Connect vpp1 to vpp2

1. Create a veth with one end named vpp1vpp2 and the other named vpp2vpp1.
2. Connect vpp1vpp2 to vpp1.
3. Connect vpp2vpp1 to vpp2.

Action: Configure Bridge Domain on vpp1

Check to see what bridge domains already exist, and select the first bridge domain number not in use:

```
sudo vppctl -s /run/vpp/cli-vpp1.sock show bridge-domain
```

```
ID   Index   Learning   U-Forwrd   UU-Flood   Flooding   ARP-Term     BVI-Intf
0      0        off        off        off        off        off        local0
```

In the example above, there is bridge domain ID '0' already. Even though sometimes we might get feedback as below:

```
no bridge-domains in use
```

the bridge domain ID '0' still exists, where no operations are supported. For instance, if we try to add host-vpp1out and host-vpp1vpp2 to bridge domain ID 0, we will get nothing setup.

```
sudo vppctl -s /run/vpp/cli-vpp1.sock set int l2 bridge host-vpp1out 0
sudo vppctl -s /run/vpp/cli-vpp1.sock set int l2 bridge host-vpp1vpp2 0
sudo vppctl -s /run/vpp/cli-vpp1.sock show bridge-domain 0 detail
```

```
show bridge-domain: No operations on the default bridge domain are supported
```

So we will create bridge domain 1 instead of playing with the default bridge domain ID 0.

Add host-vpp1out to bridge domain ID 1

```
sudo vppctl -s /run/vpp/cli-vpp1.sock set int l2 bridge host-vpp1out 1
```

Add host-vpp1vpp2 to bridge domain ID1

```
sudo vppctl -s /run/vpp/cli-vpp1.sock set int l2 bridge host-vpp1vpp2  1
```

Examine bridge domain 1:

```
sudo vppctl -s /run/vpp/cli-vpp1.sock show bridge-domain 1 detail
```

```
BD-ID   Index   BSN  Age(min)  Learning  U-Forwrd  UU-Flood  Flooding  ARP-Term  BVI-
↪Intf
  1       1       0     off        on        on        on        on       off       N/A

          Interface           If-idx ISN  SHG  BVI  TxFlood        VLAN-Tag-Rewrite
        host-vpp1out             1     1    0    -      *                none
        host-vpp1vpp2            2     1    0    -      *                none
```

Action: Configure loopback interface on vpp2

```
sudo vppctl -s /run/vpp/cli-vpp2.sock create loopback interface
```

```
loop0
```

Add the ip address 10.10.1.2/24 to vpp2 interface loop0. Set the state of interface loop0 on vpp2 to 'up'

Action: Configure bridge domain on vpp2

Check to see the first available bridge domain ID (it will be 1 in this case)

Add interface loop0 as a bridge virtual interface (bvi) to bridge domain 1

```
sudo vppctl -s /run/vpp/cli-vpp2.sock set int l2 bridge loop0 1 bvi
```

Add interface vpp2vpp1 to bridge domain 1

```
sudo vppctl -s /run/vpp/cli-vpp2.sock set int l2 bridge host-vpp2vpp1  1
```

Examine the bridge domain and interfaces.

Action: Ping from host to vpp and vpp to host

1. Add trace on vpp1 and vpp2
2. ping from host to 10.10.1.2
3. Examine and clear trace on vpp1 and vpp2
4. ping from vpp2 to 10.10.1.1

5. Examine and clear trace on vpp1 and vpp2

Action: Examine l2 fib

```
sudo vppctl -s /run/vpp/cli-vpp1.sock show l2fib verbose
```

```
    Mac Address     BD Idx           Interface           Index  static  filter  bvi  ␣
↪Mac Age (min)
 de:ad:00:00:00:00    1            host-vpp1vpp2          2       0       0     0  ␣
↪  disabled
 c2:f6:88:31:7b:8e    1            host-vpp1out           1       0       0     0  ␣
↪  disabled
2 l2fib entries
```

```
sudo vppctl -s /run/vpp/cli-vpp2.sock show l2fib verbose
```

```
    Mac Address     BD Idx           Interface           Index  static  filter  bvi  ␣
↪Mac Age (min)
 de:ad:00:00:00:00    1              loop0               2       1       0     1  ␣
↪  disabled
 c2:f6:88:31:7b:8e    1            host-vpp2vpp1          1       0       0     0  ␣
↪  disabled
2 l2fib entries
```

4.1.15 Source NAT

Skills to be Learned

1. Abusing networks namespaces for fun and profit
2. Configuring snat address
3. Configuring snat inside and outside interfaces

vpp command learned in this exercise

1. snat add interface address
2. set interface snat

Topology

Initial state

Unlike previous exercises, for this one you want to start tabula rasa.

Note: You will lose all your existing config in your vpp instances!

To clear existing config from previous exercises run:

```
ps -ef | grep vpp | awk '{print $2}'| xargs sudo kill
sudo ip link del dev vpp1host
sudo ip link del dev vpp1vpp2
```

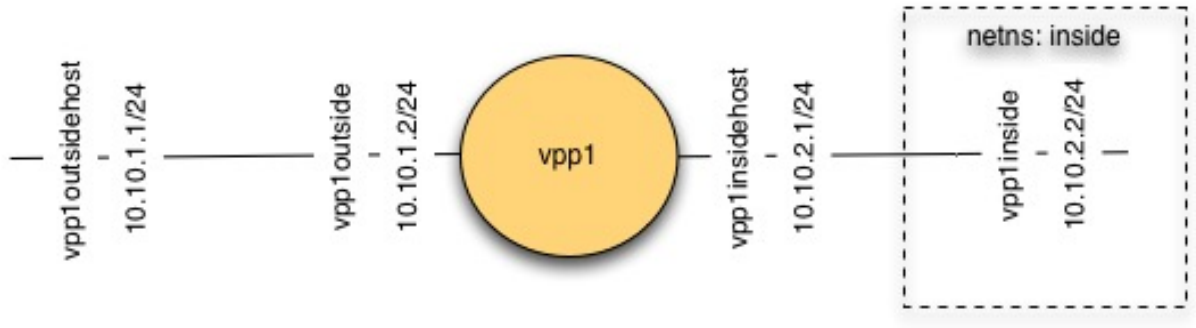

Fig. 5: SNAT Topology

Action: Install vpp-plugins

Snat is supported by a plugin, so vpp-plugins need to be installed

```
sudo apt-get install vpp-plugins
```

Action: Create vpp instance

Create one vpp instance named vpp1.

Confirm snat plugin is present:

```
sudo vppctl -s /run/vpp/cli-vpp1.sock show plugins
```

```
Plugin path is: /usr/lib/vpp_plugins
Plugins loaded:
 1.ioam_plugin.so
 2.ila_plugin.so
 3.acl_plugin.so
 4.flowperpkt_plugin.so
 5.snat_plugin.so
 6.libsixrd_plugin.so
 7.lb_plugin.so
```

Action: Create veth interfaces

1. Create a veth interface with one end named vpp1outside and the other named vpp1outsidehost
2. Assign IP address 10.10.1.1/24 to vpp1outsidehost
3. Create a veth interface with one end named vpp1inside and the other named vpp1insidehost
4. Assign IP address 10.10.2.1/24 to vpp1outsidehost

Because we'd like to be able to route *via* our vpp instance to an interface on the same host, we are going to put vpp1insidehost into a network namespace

Create a new network namespace 'inside'

```
sudo ip netns add inside
```

Move interface vpp1inside into the 'inside' namespace:

```
sudo ip link set dev vpp1insidehost up netns inside
```

Assign an ip address to vpp1insidehost

```
sudo ip netns exec inside ip addr add 10.10.2.1/24 dev vpp1insidehost
```

Create a route inside the netns:

```
sudo ip netns exec inside ip route add 10.10.1.0/24 via 10.10.2.2
```

Action: Configure vpp outside interface

1. Create a vpp host interface connected to vpp1outside
2. Assign ip address 10.10.1.2/24
3. Create a vpp host interface connected to vpp1inside
4. Assign ip address 10.10.2.2/24

Action: Configure snat

Configure snat to use the address of host-vpp1outside

```
sudo vppctl -s /run/vpp/cli-vpp1.sock snat add interface address host-vpp1outside
```

Configure snat inside and outside interfaces

```
sudo vppctl -s /run/vpp/cli-vpp1.sock set interface snat in host-vpp1inside out host-
↪vpp1outside
```

Action: Prepare to Observe Snat

Observing snat in this configuration is interesting. To do so, vagrant ssh a second time into your VM and run:

```
sudo tcpdump -s 0 -i vpp1outsidehost
```

Also enable tracing on vpp1

Action: Ping via snat

```
sudo ip netns exec inside ping -c 1 10.10.1.1
```

Action: Confirm snat

Examine the tcpdump output and vpp1 trace to confirm snat occurred.

CHAPTER 5

Reference

Note: To Do