
Vector Packet Processor Documentation

Release 0.1

John DeNisco

Aug 02, 2018

Contents

1	Overview	3
1.1	What is VPP?	3
1.2	Features	5
1.3	Performance	10
1.4	Architectures and Operating Systems	12
2	Getting Started	13
2.1	Users	13
2.2	Developers	51
2.3	Writing VPP Documentation	77
3	Use Cases	99
3.1	FD.io VPP with Containers	99
3.2	FD.io VPP with Virtual Machines	106
3.3	Using VPP as a Home Gateway	114
3.4	vSwitch/vRouter	118
4	Troubleshooting	119
4.1	How to Report an Issue	119
4.2	CPU Load/Usage	122
5	User Guides	125
5.1	Progressive VPP Tutorial	125
5.2	API User Guides	149
6	Events	151
6.1	Conferences	151
6.2	Summits	153
6.3	Meetings	163
6.4	Calls	165
6.5	Fd.io Training Event	165
7	Reference	173
7.1	VM's with Vagrant	173
7.2	Command Line Reference	179

This is beta VPP Documentation it is not meant to be complete or accurate yet!!!!

FD.io Vector Packet Processing (VPP) is a fast, scalable and multi-platform network stack.

FD.io VPP is, at it's core, a scalable layer 2-4 network stack. It supports integration into both Open Stack and Kubernetes environments. It supports network management features including configuration, counters and sampling. It supports extending with plugins, tracing and debugging. It supports use cases such as vSwitch, vRouter, Gateways, Firewalls and Load Balancers, to name but a few. Finally it is useful both a software development kit or an appliance out of the box.

1.1 What is VPP?

FD.io's Vector Packet Processing (VPP) technology is a *Fast, Scalable and Deterministic, Packet Processing* stack that runs on commodity CPUs. It provides out-of-the-box production quality switch/router functionality and much, much more. FD.io VPP is at the same time, an *Extensible and Modular Design* and *Developer Friendly* framework, capable of boot-strapping the development of packet-processing applications. The benefits of FD.io VPP are its high performance, proven technology, its modularity and flexibility, integrations and rich feature set.

FD.io VPP is vector packet processing software, to learn more about what that means, see the [what-is-vector-packet-processing](#) section.

For more detailed information on FD.io features, see the following sections:

1.1.1 Packet Processing

- Layer 2 - 4 Network Stack
 - Fast lookup tables for routes, bridge entries
 - Arbitrary n-tuple classifiers
 - Control Plane, Traffic Management and Overlays
- [Linux](#) and [FreeBSD](#) support
 - Wide support for standard Operating System Interfaces such as AF_Packet, Tun/Tap & Netmap.
- Wide network and cryptographic hardware support with [DPDK](#).
- Container and Virtualization support
 - Para-virtualized interfaces; Vhost and Virtio
 - Network Adapters over PCI passthrough
 - Native container interfaces; MemIF

- Universal Data Plane: one code base, for many use cases
 - Discrete appliances; such as [Routers](#) and [Switches](#).
 - [Cloud Infrastructure](#) and [Virtual Network Functions](#)
 - [Cloud Native Infrastructure](#)
 - The same binary package for all use cases.
- Out of the box production quality, with thanks to [CSIT](#).

For more information, please see [Features](#) for the complete list.

1.1.2 Fast, Scalable and Deterministic

- [Continuous integration and system testing](#)
 - Including continuous & extensive, latency and throughput testing
- Layer 2 Cross Connect (L2XC), typically achieve 15+ Mpps per core.
- Tested to achieve **zero** packet drops and ~15µs latency.
- Performance scales linearly with core/thread count
- Supporting millions of concurrent lookup tables entries

Please see [Performance](#) for more information.

1.1.3 Developer Friendly

- Extensive runtime counters; throughput, [instructions per cycle](#), errors, events etc.
- Integrated pipeline tracing facilities
- Multi-language API bindings
- Integrated command line for debugging
- Fault-tolerant and upgradable
 - Runs as a standard user-space process for fault tolerance, software crashes seldom require more than a process restart.
 - Improved fault-tolerance and upgradability when compared to running similar packet processing in the kernel, software updates never require system reboots.
 - Development experience is easier compared to similar kernel code
 - Hardware isolation and protection ([iommu](#))
- Built for security
 - Extensive white-box testing
 - Image segment base address randomization
 - Shared-memory segment base address randomization
 - Stack bounds checking
 - Static analysis with [Coverity](#)

1.1.4 Extensible and Modular Design

- Pluggable, easy to understand & extend
- Mature graph node architecture
- Full control to reorganize the pipeline
- Fast, plugins are equal citizens

Modular, Flexible, and Extensible

The FD.io VPP packet processing pipeline is decomposed into a ‘packet processing graph’. This modular approach means that anyone can ‘plugin’ new graph nodes. This makes VPP easily extensible and means that plugins can be customized for specific purposes. VPP is also configurable through its Low-Level API.

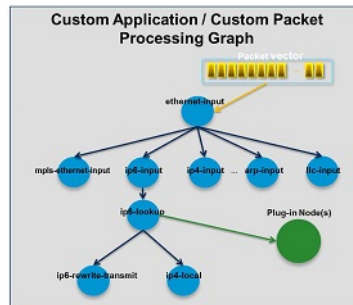


Fig. 1: Extensible and modular graph node architecture.

At runtime, the FD.io VPP platform assembles a vector of packets from RX rings, typically up to 256 packets in a single vector. The packet processing graph is then applied, node by node (including plugins) to the entire packet vector. The received packets typically traverse the packet processing graph nodes in the vector, when the network processing represented by each graph node is applied to each packet in turn. Graph nodes are small and modular, and loosely coupled. This makes it easy to introduce new graph nodes and rewire existing graph nodes.

Plugins are [shared libraries](#) and are loaded at runtime by VPP. VPP find plugins by searching the plugin path for libraries, and then dynamically loads each one in turn on startup. A plugin can introduce new graph nodes or rearrange the packet processing graph. You can build a plugin completely independently of the FD.io VPP source tree, which means you can treat it as an independent component.

1.2 Features

SDN & Cloud Integrations		Control Plane	Plugins
Tunnels	Layer 4	Traffic Management	
	Layer 3		
	Layer 2		
Devices			

1.2.1 Devices

Hardware

- [DPDK](#)

- Network Interfaces
 - Cryptographic Devices
- Open Data Plane
- Intel Ethernet Adaptive Virtual Function

Operating System

- Netmap
- af_packet
- Tap V2 (FastTap)

Virtualization:

- SSVM
- Vhost / VirtIO

Containers

- Vhost-user
- MemIF

1.2.2 SDN & Cloud Integrations

1.2.3 Traffic Management

IP Layer Input Checks

- Source Reverse Path Forwarding
- Time To Live expiration
- IP header checksum
- Layer 2 Length < IP Length

Classifiers

- Multiple million Classifiers - Arbitrary N-tuple

Policers

- Colour Aware & Token Bucket
- Rounding Closest/Up/Down
- Limits in PPS/KBPS
- Types:

- Single Rate Two Colour
 - Single Rate Three Colour
 - Dual Rate Three Colour
- Action Triggers
 - Conform
 - Exceed
 - Violate
- Actions Type
 - Drop
 - Transmit
 - Mark-and-transmit

Switched Port Analyzer (SPAN) * mirror traffic to another switch port

ACLs

- Stateful
- Stateless

COP

MAC/IP Pairing

(security feature).

1.2.4 Layer 2

MAC Layer

- Ethernet

Discovery

- Cisco Discovery Protocol
- Link Layer Discovery Protocol (LLDP)

Link Layer Control Protocol

- Bit Index Explicit Replication – Link Layer Multi-cast forwarding.
- Link Layer Control (LLC) - multiplex protocols over the MAC layer.
- Spatial Reuse Protocol (SRP)
- High-Level Data Link Control (HDLC)
- Logical link control (LLC)

- Link Agg Control Protocol (Active/Active, Active/Passive) – 18.04

Virtual Private Networks

- MPLS
 - MPLS-o-Ethernet – Deep label stacks supported
- Virtual Private LAN Service (VPLS)
- VLAN
- Q-in-Q
- Tag-rewrite (VTR) - push/pop/Translate (1:1,1:2, 2:1,2:2)
- Ethernet flow point Filtering
- Layer 2 Cross Connect

Bridging

- Bridge Domains
- MAC Learning (50k addresses)
- Split-horizon group support
- Flooding

ARP

- Proxy
- Termination
- Bidirectional Forwarding Detection

Integrated Routing and Bridging (IRB)

- Flexibility to both route and switch between groups of ports.
- Bridged Virtual Interface (BVI) Support, allows traffic switched traffic to be routed.

1.2.5 Layer 3

IP Layer

- ICMP
- IPv4
- IPv6
- IPSEC
- Link Local Addressing

MultiCast

- Multicast FiB
- IGMP

Virtual Routing and forwarding (VRF)

- VRF scaling, thousands of tables.
- Controlled cross-VRF lookups

Multi-path

- Equal Cost Multi Path (ECMP)
- Unequal Cost Multi Path (UCMP)

IPv4

- ARP
- ARP Proxy
- ARP Snooping

IPv6

- Neighbour discovery (ND)
- ND Proxy
- Router Advertisement
- Segment Routing
- Distributed Virtual Routing Resolution

Forwarding Information Base (FIB)

- Hierarchical FIB
- Memory efficient
- Multi-million entry scalable
- Lockless/concurrent updates
- Recursive lookups
- Next hop failure detection
- Shared FIB adjacencies
- Multicast support
- MPLS support

1.2.6 Layer 4

1.2.7 Tunnels

Layer 2

- L2TP
- PPP
- VLAN

Layer 3

- Mapping of Address and Port with Encapsulation (MAP-E)
- Lightweight IPv4 over IPv6
 - An Extension to the Dual-Stack Lite Architecture
- GENEVE
- VXLAN

Segment Routing

- IPv6
- MPLS

Generic Routing Encapsulation (GRE) * GRE over IPSEC * GRE over IP * MPLS * NSH

1.2.8 Control Plane

- DHCP client/proxy
- DHCPv6 Proxy

1.2.9 Plugins

- iOAM

1.3 Performance

1.3.1 Overview

One of the benefits of FD.io VPP, is high performance on relatively low-power computing, this performance is based on the following features:

- A high-performance user-space network stack designed for commodity hardware.
 - L2, L3 and L4 features and encapsulations.
- Optimized packet interfaces supporting a multitude of use cases.

- An integrated vhost-user backend for high speed VM-to-VM connectivity.
- An integrated memif container backend for high speed Container-to-Container connectivity.
- An integrated vhost based interface to punt packets to the Linux Kernel.
- The same optimized code-paths run execute on the host, and inside VMs and Linux containers.
- Leverages best-of-breed open source driver technology: [DPDK](#).
- Tested at scale; linear core scaling, tested with millions of flows and mac addresses.

These features have been designed to take full advantage of common micro-processor optimization techniques, such as:

- Reducing cache and TLS misses by processing packets in vectors.
- Realizing [IPC](#) gains with vector instructions such as: SSE, AVX and NEON.
- Eliminating mode switching, context switches and blocking, to always be doing useful work.
- Cache-lined aligned buffers for cache and memory efficiency.

1.3.2 Packet Throughput Graphs

These are some of the packet throughput graphs for FD.io VPP 18.04 from the CSIT [18.04 benchmarking report](#).

L2 Ethernet Switching Throughput Tests

VPP NDR 64B packet throughput in 1 Core, 1 Thread setup, is presented in the graph below.

NDR Performance Tests

This is a VPP NDR 64B packet throughput in 1 Core, 1 Thread setup, live graph of the NDR (No Drop Rate) L2 Performance Tests.

IPv4 Routed-Forwarding Performance Tests

VPP NDR 64B packet throughput in 1t1c setup (1thread, 1core) is presented in the graph below.

IPv6 Routed-Forwarding Performance Tests

VPP NDR 78B packet throughput in 1t1c setup (1 thread, 1 core) is presented in the graph below.

1.3.3 Trending Throughput Graphs

These are some of the trending packet throughput graphs from the CSIT [trending dashboard](#). **Please note that,** performance in the trending graphs will change on a nightly basis in line with the software development cycle.

L2 Ethernet Switching Performance Tests

This is a live graph of the 1 Core, 1 Thread, L2 Ethernet Switching Performance Tests Test on the x520 NIC.

IPv4 Routed-Forwarding Performance Tests

This is a live graph of the IPv4 Routed Forwarding Switching Performance Tests.

IPv6 Routed-Forwarding Performance Tests

VPP NDR 78B packet throughput in 1t1c setup (1 thread, 1 core) is presented in the trending graph below.

1.3.4 For More information on CSIT

These are FD.io Continuous System Integration and Testing (CSIT)'s documentation links.

- [CSIT Code Documentation](#)
- [CSIT Test Overview](#)
- [VPP Performance Dashboard](#)

1.4 Architectures and Operating Systems

1.4.1 Architectures

- – The FD.io VPP platform supports:
 - * x86/64
 - * ARM

1.4.2 Operating Systems and Packaging

FD.io VPP supports package installation on the following recent LTS operating systems releases:

- – Operating Systems:
 - * Debian
 - * Ubuntu
 - * CentOS
 - * OpenSUSE

2.1 Users

2.1.1 Installing VPP from Packages

If you are simply using vpp, it can be convenient to install the binaries from existing packages. This guide will describe how pull, install and run the VPP packages.

Package Descriptions

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

Packages

vpp

Vector Packet Processing executables. This is the primary package that must be installed to use VPP. This package contains:

- 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. The 'vpp' package depends on this package, so it will always be installed. 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.

vpp-selinux-policy

This package contains the VPP Custom SELinux Policy. It is only generated for Fedora and CentOS distros. For those distros, the 'vpp' package depends on this package, so it will always be installed. It will not enable SELinux on the system. It will install a Custom VPP SELinux policy that will be used if SELinux is enabled at any time.

Installing VPP Binaries

Installing on Ubuntu

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

Ubuntu 16.04 - Setup the fd.io Repository

From the following choose one of the releases to install.

Update the OS

It is probably a good idea to update and upgrade the OS before starting

```
apt-get update
```

Point to the Repository

Create a file “**/etc/apt/sources.list.d/99fd.io.list**” with the contents that point to the version needed. The contents needed are shown below.

VPP latest Release

Create the file **/etc/apt/sources.list.d/99fd.io.list** with contents:

```
deb [trusted=yes] https://nexus.fd.io/content/repositories/fd.io.ubuntu.xenial.main/ .  
↪/
```

VPP stable/1804 Branch

Create the file **/etc/apt/sources.list.d/99fd.io.list** with contents:

```
deb [trusted=yes] https://nexus.fd.io/content/repositories/fd.io.stable.1804.ubuntu.  
↪xenial.main/ ./
```

VPP master Branch

Create the file `/etc/apt/sources.list.d/99fd.io.list` with contents:

```
deb [trusted=yes] https://nexus.fd.io/content/repositories/fd.io.master.ubuntu.xenial.  
↪main/ ./
```

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*
```

Installing on Centos

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

Setup the fd.io Repository - Centos 7

Update the OS

It is probably a good idea to update and upgrade the OS before starting:

```
$ sudo yum update
```

Point to the Repository

For CentOS based systems, there are two repositories to pull VPP binaries from.

- CentOS NFV SIG Repository
- Nexus Repository

CentOS NFV SIG Repository

VPP is not in the official CentOS 7 distro. However, CentOS has Special Interest Groups (SIG), which are smaller groups within the CentOS community that focus on a small set of issues. The CentOS NFV (Network Function

Virtualization) SIG was created to provide a CentOS-based stack that will serve as a platform for the deployment and testing of virtual network functions (VNFs). VPP has been included in this SIG.

To install released packages from the CentOS NFV SIG Repository on an updated Centos 7 system, first, install the CentOS NFV SIG FIDO repo file:

```
$ sudo yum install centos-release-fdio
```

then **‘Install VPP RPMs’**, as described below.

This will install the latest VPP version from the repository. To install an older version, once the CentOS NFV SIG FDIO repo file has been installed, list the stored versions:

```
$ sudo yum --showduplicates list vpp* | expand
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: repos-va.psychz.net
 * epel: download-ib01.fedoraproject.org
 * extras: mirror.siena.edu
 * updates: repol.ash.innoscale.net
Available Packages
vpp.x86_64                17.10-1                centos-fdio
vpp.x86_64                18.01.1-1              centos-fdio
vpp.x86_64                18.01.2-1              centos-fdio
vpp.x86_64                18.04-1                centos-fdio
vpp-api-java.x86_64      17.10-1                centos-fdio
vpp-api-java.x86_64      18.01.1-1              centos-fdio
vpp-api-java.x86_64      18.01.2-1              centos-fdio
vpp-api-java.x86_64      18.04-1                centos-fdio
vpp-api-lua.x86_64       17.10-1                centos-fdio
vpp-api-lua.x86_64       18.01.1-1              centos-fdio
vpp-api-lua.x86_64       18.01.2-1              centos-fdio
vpp-api-lua.x86_64       18.04-1                centos-fdio
vpp-api-python.x86_64    17.10-1                centos-fdio
vpp-api-python.x86_64    18.01.1-1              centos-fdio
vpp-api-python.x86_64    18.01.2-1              centos-fdio
vpp-api-python.x86_64    18.04-1                centos-fdio
vpp-devel.x86_64         17.10-1                centos-fdio
vpp-devel.x86_64         18.01.1-1              centos-fdio
vpp-devel.x86_64         18.01.2-1              centos-fdio
vpp-devel.x86_64         18.04-1                centos-fdio
vpp-lib.x86_64           17.10-1                centos-fdio
vpp-lib.x86_64           18.01.1-1              centos-fdio
vpp-lib.x86_64           18.01.2-1              centos-fdio
vpp-lib.x86_64           18.04-1                centos-fdio
vpp-plugins.x86_64       17.10-1                centos-fdio
vpp-plugins.x86_64       18.01.1-1              centos-fdio
vpp-plugins.x86_64       18.01.2-1              centos-fdio
vpp-plugins.x86_64       18.04-1                centos-fdio
vpp-selinux-policy.x86_64 18.04-1                centos-fdio
```

Then install a particular version:

```
$ sudo yum install vpp-17.10-1.x86_64
```

Nexus Repository

Build artifacts are also posted to a FD.io Nexus Repository. This includes official point releases, as well as nightly builds. To use any of these build artifacts, create a file `/etc/yum.repos.d/fdio-release.repo` with the content that points to the version needed. Below are some common examples of the content needed:

VPP Latest Release

To allow `'yum'` access to the official VPP releases, create the file `/etc/yum.repos.d/fdio-release.repo` with the following content:

```
$ cat /etc/yum.repos.d/fdio-release.repo
[fdio-release]
name=fd.io release branch latest merge
baseurl=https://nexus.fd.io/content/repositories/fd.io.centos7/
enabled=1
gpgcheck=0
```

The `'yum install vpp'` command will install the most recent release. To install older releases, run the following command to get the list of releases provided:

```
$ sudo yum --showduplicates list vpp* | expand
```

Then choose the release to install. See **'CentOS NFV SIG Repository'** for sample `'yum --showduplicates list'` output and an example of installing a particular version of the RPMs.

VPP Stable Branch

To allow `yum` access to the build artifacts for a VPP stable branch, create the file `/etc/yum.repos.d/fdio-release.repo` with the following content:

```
$ cat /etc/yum.repos.d/fdio-release.repo
[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
```

For other stable branches, replace the `'1804'` from the above content with the desired release. Examples: 1606, 1609, 1701, 1704, 1707, 1710, 1804, 1807

The `'yum install vpp'` command will install the most recent build on the branch, not the latest official release. Run the following command to get the list of images produced by the branch:

```
$ sudo yum --showduplicates list vpp* | expand
```

Then choose the image to install. See **'CentOS NFV SIG Repository'** for sample `'yum --showduplicates list'` output and an example of installing a particular version of the RPMs.

VPP Master Branch

To allow `yum` access to the nightly builds from the VPP master branch, create the file `/etc/yum.repos.d/fdio-release.repo` with the following content:

```
$ cat /etc/yum.repos.d/fdio-release.repo
[fdio-master]
name=fd.io master branch latest merge
baseurl=https://nexus.fdio.io/content/repositories/fdio.master.centos7/
enabled=1
gpgcheck=0
```

The `'yum install vpp'` command will install the most recent build on the branch. Run the following command to get the list of images produce by the branch:

```
$ sudo yum --showduplicates list vpp* | expand
```

Then choose the image to install. See **‘CentOS NFV SIG Repository’** for sample `'yum --showduplicates list'` output and an example of installing a particular version of the RPMs.

Install VPP RPMs

To install the VPP packet engine, run the following:

```
$ sudo yum install vpp
```

The **‘vpp’** RPM depend on the **‘vpp-lib’** and **‘vpp-selinux-policy’** RPMs, so they will be installed as well.

Note: The **‘vpp-selinux-policy’** will not enable SELinux on the system. It will install a Custom VPP SELinux policy that will be used if SELinux is enabled at any time.

There are additional packages that are optional. These packages can be combined with the command above and installed all at once, or installed as needed:

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

Starting VPP

Once VPP is installed on the system, to run VPP as a systemd service on CentOS, run:

```
$ sudo systemctl start vpp
```

Then to enable VPP to start on system reboot:

```
$ sudo systemctl enable vpp
```

Outside of running VPP as a systemd service, VPP can be started manually or made to run within GDB for debugging. See [Running VPP](#) for more details and ways to tailor VPP to a specific system.

Uninstall the VPP RPMs

```
$ sudo yum autoremove vpp*
```

Installing on openSUSE

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

Installing

To 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
```

For More Information

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

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

2.1.2 Configuring VPP

There is some basic configuration that is need to run FD.io VPP. This section will describe some basic configuration.

Huge Pages

VPP requires ‘*hugepages*’ to run. VPP will overwrite existing hugepage settings when VPP is installed. By default, VPP sets the number of hugepages on a system to 1024 2M hugepages (1G hugepages are no longer supported). This is the number of hugepages on the system, not just used by VPP. When VPP is installed, the following file is copied to the system and used to apply the hugepage settings on VPP installation and system reboot:

```
$ cat /etc/sysctl.d/80-vpp.conf
# Number of 2MB hugepages desired
vm.nr_hugepages=1024

# Must be greater than or equal to (2 * vm.nr_hugepages).
vm.max_map_count=3096

# All groups allowed to access hugepages
vm.hugetlb_shm_group=0

# Shared Memory Max must be greater or equal to the total size of hugepages.
# For 2MB pages, TotalHugepageSize = vm.nr_hugepages * 2 * 1024 * 1024
# If the existing kernel.shmmax setting (cat /sys/proc/kernel/shmmax)
# is greater than the calculated TotalHugepageSize then set this parameter
# to current shmmax value.
kernel.shmmax=2147483648
```

Depending on how the system is being used, this file can be updated to adjust the number of hugepages reserved on a system. Below are some examples of possible values.

For a small VM with minimal workload:

```
vm.nr_hugepages=512
vm.max_map_count=2048
kernel.shmmax=1073741824
```

For a large system running multiple VMs, each needing its own set of hugepages:

```
vm.nr_hugepages=32768
vm.max_map_count=66560
kernel.shmmax=68719476736
```

Note: If VPP is being run in a Virtual Machine (VM), the VM must have hugepage backing. When VPP is installed, it will attempt to overwrite existing hugepage setting. If the VM does not have hugepage backing, this will fail, but this may go unnoticed. When the VM is rebooted, on system startup, ‘*vm.nr_hugepages*’ will be reapplied, will fail, and the VM will abort kernel boot, locking up the VM. To avoid this scenario, ensure the VM has enough hugepage backing.

VPP Configuration File - ‘*startup.conf*’

After a successful installation, VPP installs a startup config file named ‘*startup.conf*’ in the ‘*/etc/vpp/*’ directory. This file can be tailored to make VPP run as desired, but contains default values for typical installations. Below are more details about this file and parameter and values it contains.

Introduction

The VPP network stack comes with several configuration options that can be provided either on the command line when VPP is started, or in a configuration file. Specific applications built on the stack have been known to require a dozen arguments, depending on requirements.

Command-line Arguments

Parameters are grouped by a section name. When providing more than one parameter to a section, all parameters for that section must be wrapped in curly braces. For example, to start VPP with configuration data via the command line with the section name *'unix'*:

```
$ sudo /usr/bin/vpp unix { interactive cli-listen 127.0.0.1:5002 }
```

The command line can be presented as a single string or as several; anything given on the command line is concatenated with spaces into a single string before parsing. VPP applications must be able to locate their own executable images. The simplest way to ensure this will work is to invoke a VPP application by giving its absolute path. For example: *'/usr/bin/vpp <options>'* At startup, VPP applications parse through their own ELF-sections [primarily] to make lists of init, configuration, and exit handlers.

When developing with VPP, in gdb it's often sufficient to start an application like this:

```
(gdb) run unix interactive
```

Configuration File

It is also possible to supply the configuration parameters in a startup configuration. The path of the file is provided to the VPP application on its command line. The format of the configuration file is a simple text file with the same content as the command line, but with the benefit of being able to use newlines to make the content easier to read. For example:

```
$ cat /etc/vpp/startup.conf
unix {
    nodaemon
    log /var/log/vpp/vpp.log
    full-coredump
    cli-listen localhost:5002
}

api-trace {
    on
}

dpdk {
    dev 0000:03:00.0
}
```

VPP is then instructed to load this file with the *-c* option. For example:

```
$ sudo /usr/bin/vpp -c /etc/vpp/startup.conf
```

When the VPP service is started, VPP is started with this option via another installed file, *vpp.service* (Ubuntu: */lib/systemd/system/vpp.service* and CentOS: */usr/lib/systemd/system/vpp.service*). See *'ExecStart'* below:

```
$ cat /lib/systemd/system/vpp.service
[Unit]
Description=vector packet processing engine
After=network.target

[Service]
Type=simple
ExecStartPre=/bin/rm -f /dev/shm/db /dev/shm/global_vm /dev/shm/vpe-api
ExecStartPre=/sbin/modprobe uio_pci_generic
ExecStart=/usr/bin/vpp -c /etc/vpp/startup.conf
ExecStopPost=/bin/rm -f /dev/shm/db /dev/shm/global_vm /dev/shm/vpe-api
Restart=always

[Install]
WantedBy=multi-user.target
```

Configuration Parameters

Below is the list of section names and their associated parameters. This is not an exhaustive list of parameters available. The command-line argument parsers can be found in the source code by searching for instances of the **VLIB_CONFIG_FUNCTION** and **VLIB_EARLY_CONFIG_FUNCTION** macro.

For example, the invocation '*VLIB_CONFIG_FUNCTION (foo_config, "foo")*' will cause the function '*foo_config*' to receive all parameters given in a parameter block named "foo": "foo { arg1 arg2 arg3 ... }".

List of Basic Parameters:

unix
dpdk
cpu

List of Advanced Parameters:

acl-plugin
api-queue
api-segment
api-trace
buffers
cj
dns
heapsize
ip
ip6
l2learn
l2tp
logging
mactime
map
mc

nat
oam
plugins
plugin_path
punt
session
socketsvr
stats
statseg
tapcli
tcp
tls
tuntap
vhost-user
vlib

“unix” Parameters

Configure VPP startup and behavior type attributes, as well and any OS based attributes.

- **interactive** Attach CLI to stdin/out and provide a debugging command line interface. Implies nodaemon.

Example: interactive

- **nodaemon** Do not fork / background the vpp process. Typical when invoking VPP applications from a process monitor. Set by default in the default *startup.conf* file.

Example: nodaemon

- **log <filename>** Logs the startup configuration and all subsequent CLI commands in filename. Very useful in situations where folks don't remember or can't be bothered to include CLI commands in bug reports. The default *startup.conf* file is to write to */var/log/vpp/vpp.log*.

In VPP 18.04, the default log file location was moved from */tmp/vpp.log* to */var/log/vpp/vpp.log*. The VPP code is indifferent to the file location. However, if SELinux is enabled, then the new location is required for the file to be properly labeled. Check your local *startup.conf* file for the log file location on your system.

Example: log /var/log/vpp/vpp-debug.log

- **exec!startup-config <filename>** Read startup operational configuration from filename. The contents of the file will be performed as though entered at the CLI. The two keywords are aliases for the same function; if both are specified, only the last will have an effect. The file contains CLI commands, for example:

```
$ cat /usr/share/vpp/scripts/interface-up.txt
set interface state TenGigabitEthernet1/0/0 up
set interface state TenGigabitEthernet1/0/1 up
```

Example: startup-config /usr/share/vpp/scripts/interface-up.txt

- **gid number|name>** Sets the effective group ID to the input group ID or group name of the calling process.

Example: gid vpp

- **full-coredump** Ask the Linux kernel to dump all memory-mapped address regions, instead of just text+data+bss.
Example: full-coredump
- **coredump-size unlimited|<n>G|<n>M|<n>K|<n>** Set the maximum size of the coredump file. The input value can be set in GB, MB, KB or bytes, or set to *'unlimited'*.
Example: coredump-size unlimited
- **cli-listen <ipaddress:port>|<socket-path>** Bind the CLI to listen at address localhost on TCP port 5002. This will accept an ipaddress:port pair or a filesystem path; in the latter case a local Unix socket is opened instead. The default *'startup.conf'* file is to open the socket *'/run/vpp/cli.sock'*.
Example: cli-listen localhost:5002 **Example:** cli-listen /run/vpp/cli.sock
- **cli-line-mode** Disable character-by-character I/O on stdin. Useful when combined with, for example, emacs M-x gud-gdb.
Example: cli-line-mode
- **cli-prompt <string>** Configure the CLI prompt to be string.
Example: cli-prompt vpp-2
- **cli-history-limit <n>** Limit command history to <n> lines. A value of 0 disables command history. Default value: 50
Example: cli-history-limit 100
- **cli-no-banner** Disable the login banner on stdin and Telnet connections.
Example: cli-no-banner
- **cli-no-pager** Disable the output pager.
Example: cli-no-pager
- **cli-pager-buffer-limit <n>** Limit pager buffer to <n> lines of output. A value of 0 disables the pager. Default value: 100000
Example: cli-pager-buffer-limit 5000
- **runtime-dir <dir>** Set the runtime directory, which is the default location for certain files, like socket files. Default is based on User ID used to start VPP. Typically it is *'root'*, which defaults to *'/run/vpp/'*. Otherwise, defaults to *'/run/user/<uid>/vpp/'*.
Example: runtime-dir /tmp/vpp
- **poll-sleep-usec <n>** Add a fixed-sleep between main loop poll. Default is 0, which is not to sleep.
Example: poll-sleep-usec 100
- **pidfile <filename>** Writes the pid of the main thread in the given filename.
Example: pidfile /run/vpp/vpp1.pid

“dpdk” Parameters

Command line DPDK configuration controls a number of parameters, including device whitelisting, the number of CPUs available for launching dpdk-eal-controlled threads, the number of I/O buffers, and the process affinity mask. In addition, the DPDK configuration function attempts to support all of the DPDK EAL configuration parameters.

All of the DPDK EAL options should be available. See *../src/plugins/dpdk/device/dpdk_priv.h*, look at the set of “foreach_eal_XXX” macros.

Popular options include:

- **dev <pci-dev>** White-list [as in, attempt to drive] a specific PCI device. PCI-dev is a string of the form “DDDD:BB:SS.F” where:

DDDD = Domain
BB = Bus Number
SS = Slot number
F = Function

This is the same format used in the linux sysfs tree (i.e. /sys/bus/pci/devices) for PCI device directory names.

Example: dev 0000:02:00.0

- **dev <pci-dev> { .. }** When whitelisting specific interfaces by specifying PCI address, additional custom parameters can also be specified. Valid options include:
 - **num-rx-queues <n>** Number of receive queues. Also enables RSS. Default value is 1.
 - **num-tx-queues <n>** Number of transmit queues. Default is equal to number of worker threads or 1 if no workers threads.
 - **num-rx-desc <n>** Number of descriptors in receive ring. Increasing or reducing number can impact performance. Default is 1024.
 - **num-rt-desc <n>** Number of descriptors in transmit ring. Increasing or reducing number can impact performance. Default is 1024.
 - **workers** TBD
 - **vlan-strip-offload on/off:** VLAN strip offload mode for interface. VLAN stripping is off by default for all NICs except VICs, using ENIC driver, which has VLAN stripping on by default.
 - **hqos** Enable the Hierarchical Quality-of-Service (HQoS) scheduler, default is disabled. This enables HQoS on specific output interface.
 - **hqos { .. }** HQoS can also have its own set of custom parameters. Setting a custom parameter also enables HQoS.
 - * **hqos-thread <n>** HQoS thread used by this interface. To setup a pool of threads that are shared by all HQoS interfaces, set via the ‘*cpu’* section using either ‘corelist-hqos-threads’ or ‘coremask-hqos-threads’.
 - **rss** TBD

Example:

```
dev 0000:02:00.1 {  
    num-rx-queues 2  
    num-tx-queues 2  
}
```

- **vdev <eal-command>** Provide a DPDK EAL command to specify bonded Ethernet interfaces, operating modes and PCI addresses of slave links. Only XOR balanced (mode 2) mode is supported.

Example:

```
vdev eth_bond0,mode=2,slave=0000:0f:00.0,slave=0000:11:00.0,xmit_policy=l34  
vdev eth_bond1,mode=2,slave=0000:10:00.0,slave=0000:12:00.0,xmit_policy=l34
```

- **num-mbufs <n>** Increase number of buffers allocated. May be needed in scenarios with large number of interfaces and worker threads, or a lot of physical interfaces with multiple RSS queues. Value is per CPU socket. Default is 16384.

Example: num-mbufs 128000

- **no-pci** When VPP is started, if an interface is not owned by the linux kernel (interface is administratively down), VPP will attempt to manage the interface. *'no-pci'* indicates that VPP should not walk the PCI table looking for interfaces.

Example: no-pci

- **no-hugetlb** Don't use huge TLB pages. Potentially useful for running simulator images.

Example: no-hugetlb

- **kni <n>** Number of KNI interfaces. Refer to the DPDK documentation.

Example: kni 2

- **uio-driver uio_pci_generic|igb_uio|vfio-pci|auto** Change UIO driver used by VPP. Default is *'auto'*.

Example: uio-driver igb_uio

- **socket-mem <n>** Change hugepages allocation per-socket, needed only if there is need for larger number of mbufs. Default is 64 hugepages on each detected CPU socket.

Example: socket-mem 2048,2048

Other options include:

- **enable-tcp-udp-checksum** Enables UDP/TCP RX checksum offload.

Example: enable-tcp-udp-checksum

- **no-multi-seg** Disable multi-segment buffers, improves performance but disables Jumbo MTU support.

Example: no-multi-seg

- **no-tx-checksum-offload** Disables UDP/TCP TX checksum offload. Typically needed for use faster vector PMDs (together with no-multi-seg).

Example: no-tx-checksum-offload

- **decimal-interface-names** Format DPDK device names with decimal, as opposed to hexadecimal.

Example: decimal-interface-names

- **log-level emergency|alert|critical|error|warning|notice|info|debug** Set the log level for DPDK logs. Default is *'notice'*.

Example: log-level error

- **dev default { .. }** Change default settings for all interfaces. This section supports the same set of custom parameters described in *'dev <pci-dev> { .. }'*.

Example:

```
dev default {  
    num-rx-queues 3  
    num-tx-queues 3  
}
```

“cpu” Parameters

Command-line CPU configuration controls the creation of named thread types, and the cpu affinity thereof. In the VPP there is one main thread and optionally the user can create worker(s). The main thread and worker thread(s) can be pinned to CPU core(s) automatically or manually.

Automatic Pinning:

- **workers <n>** Create <n> worker threads.

Example: workers 4

- **io <n>** Create <n> i/o threads.

Example: io 2

- **main-thread-io** Handle i/o devices from thread 0, hand off traffic to worker threads. Requires “workers <n>”.

Example: main-thread-io

- **skip-cores <n>** Sets number of CPU core(s) to be skipped (1 ... N-1). Skipped CPU core(s) are not used for pinning main thread and working thread(s). The main thread is automatically pinned to the first available CPU core and worker(s) are pinned to next free CPU core(s) after core assigned to main thread. Leave the low nn bits of the process affinity mask clear.

Example: skip-cores 4

Manual Pinning:

- **main-core <n>** Assign main thread to a specific core.

Example: main-core 1

- **coremask-workers <hex-mask>** Place worker threads according to the bitmap hex-mask.

Example: coremask-workers 0x0000000000C0000C

- **corelist-workers <list>** Same as coremask-workers but accepts a list of cores instead of a bitmap.

Example: corelist-workers 2-3,18-19

- **coremask-io <hex-mask>** Place I/O threads according to the bitmap hex-mask.

Example: coremask-io 0x0000000003000030

- **corelist-io <list>** Same as coremask-io but accepts a list of cores instead of a bitmap.

Example: corelist-io 4-5,20-21

- **coremask-hqos-threads <hex-mask>** Place HQoS threads according to the bitmap hex-mask. A HQoS thread can run multiple HQoS objects each associated with different output interfaces.

Example: coremask-hqos-threads 0x000000000C0000C0

- **corelist-hqos-threads <list>** Same as coremask-hqos-threads but accepts a list of cores instead of a bitmap.

Example: corelist-hqos-threads 6-7,22-23

Other:

- **use-pthreads** TBD

Example: use-pthreads

- **thread-prefix <prefix>** Set a prefix to be prepended to each thread name. The thread name already contains an underscore. If not provided, the default is ‘vpp’. Currently, prefix used on threads: ‘vpp_main’, ‘vpp_stats’

Example: thread-prefix vpp1

- **scheduler-policy** *rr|fifo|batch|idle|other* TBD

Example: scheduler-policy fifo

- **scheduler-priority** *<n>* Set the scheduler priority. Only valid if the ‘*scheduler-policy*’ is set to ‘*fifo*’ or ‘*rr*’. The valid ranges for the scheduler priority depends on the ‘*scheduler-policy*’ and the current kernel version running. The range is typically 1 to 99, but see the linux man pages for ‘*sched*’ for more details. If this value is not set, the current linux kernel default is left in place.

Example: scheduler-priority 50

- **<thread-name> <count>** Set the number of threads for a given thread (by name). Some threads, like ‘*stats*’, have a fixed number of threads and cannot be changed. List of possible threads include (but not limited too): *hqos-threads*, *workers*

Example: hqos-threads 2

Note: The “main” thread always occupies the lowest core-id specified in the DPDK [process-level] coremask.

Here’s a full-bore manual placement example:

```
/usr/bin/vpp unix interactive tuntap disable cpu { main-thread-io coremask-workers_
↪18 coremask-stats 4 } dpdk { coremask 1e }

# taskset -a -p <vpe-pid>
pid 16251's current affinity mask: 2           # main thread
pid 16288's current affinity mask: ffffffff    # DPDK interrupt thread (not bound to a_
↪core)
pid 16289's current affinity mask: 4           # stats thread
pid 16290's current affinity mask: 8           # worker thread 0
pid 16291's current affinity mask: 10          # worker thread 1
```

“acl-plugin” Parameters

The following parameters should only be set by those that are familiar with the interworkings of VPP and the ACL Plugin.

The first three parameters, *connection hash buckets*, *connection hash memory*, and *connection count max*, set the **connection table per-interface parameters** for modifying how the two bounded-index extensible hash tables for IPv6 (40*8 bit key and 8*8 bit value pairs) and IPv4 (16*8 bit key and 8*8 bit value pairs) **ACL plugin FA interface sessions** are initialized.

- **connection hash buckets** *<n>* Sets the number of hash buckets (rounded up to a power of 2) in each of the two bi-hash tables. Defaults to 64*1024 (65536) hash buckets.

Example: connection hash buckets 65536

- **connection hash memory** *<n>* Sets the number of bytes used for “backing store” allocation in each of the two bi-hash tables. Defaults to 1073741824 bytes.

Example: connection hash memory 1073741824

- **connection count max** *<n>* Sets the maximum number of pool elements when allocating each per-worker pool of sessions for both bi-hash tables. Defaults to 500000 elements in each pool.

Example: connection count max 500000

- **main heap size <n>G|<n>M|<n>K|<n>** Sets the size of the main memory heap that holds all the ACL module related allocations (other than hash.) Default size is 0, but during ACL heap initialization is equal to *per_worker_size_with_slack * tm->n_vlib_mains + bihash_size + main_slack*. Note that these variables are partially based on the **connection table per-interface parameters** mentioned above.

Example: main heap size 3G

The next three parameters, *hash lookup heap size*, *hash lookup hash buckets*, and *hash lookup hash memory*, modify the initialization of the bi-hash lookup table used by the ACL plugin. This table is initialized when attempting to apply an ACL to the existing vector of ACLs looked up during packet processing (but it is found that the table does not exist / has not been initialized yet.)

- **hash lookup heap size <n>G|<n>M|<n>K|<n>** Sets the size of the memory heap that holds all the miscellaneous allocations related to hash-based lookups. Default size is 67108864 bytes.

Example: hash lookup heap size 70M

- **hash lookup hash buckets <n>** Sets the number of hash buckets (rounded up to a power of 2) in the bi-hash lookup table. Defaults to 65536 hash buckets.

Example: hash lookup hash buckets 65536

- **hash lookup hash memory <n>** Sets the number of bytes used for “backing store” allocation in the bi-hash lookup table. Defaults to 67108864 bytes.

Example: hash lookup hash memory 67108864

- **use tuple merge <n>** Sets a boolean value indicating whether or not to use TupleMerge for hash ACL’s. Defaults to 1 (true), meaning the default implementation of hashing ACL’s **does use** TupleMerge.

Example: use tuple merge 1

- **tuple merge split threshold <n>** Sets the maximum amount of rules (ACE’s) that can collide in a bi-hash lookup table before the table is split into two new tables. Splitting ensures less rule collisions by hashing colliding rules based on their common tuple (usually their maximum common tuple.) Splitting occurs when the *length of the colliding rules vector* is greater than this threshold amount. Defaults to a maximum of 39 rule collisions per table.

Example: tuple merge split threshold 30

- **reclassify sessions <n>** Sets a boolean value indicating whether or not to take the epoch of the session into account when dealing with re-applying ACL’s or changing already applied ACL’s. Defaults to 0 (false), meaning the default implementation **does NOT** take the epoch of the session into account.

Example: reclassify sessions 1

“api-queue” Parameters

The following parameters should only be set by those that are familiar with the interworkings of VPP.

- **length <n>** Sets the api queue length. Minimum valid queue length is 1024, which is also the default.

Example: length 2048

“api-segment” Parameters

These values control various aspects of the binary API interface to VPP.

- **prefix <path>** Sets the prefix prepended to the name used for shared memory (SHM) segments. The default is empty, meaning shared memory segments are created directly in the SHM directory */dev/shm*. It is

worth noting that on many systems `/dev/shm` is a symbolic link to somewhere else in the file system; Ubuntu links it to `/run/shm`.

Example: prefix `/run/shm`

- **uid <number|name>** Sets the user ID or name that should be used to set the ownership of the shared memory segments. Defaults to the same user that VPP is started with, probably root.

Example: uid root

- **gid <number|name>** Sets the group ID or name that should be used to set the ownership of the shared memory segments. Defaults to the same group that VPP is started with, probably root.

Example: gid vpp

The following parameters should only be set by those that are familiar with the interworkings of VPP.

- **baseva <x>** Set the base address for SVM global region. If not set, on AArch64, the code will try to determine the base address. All other default to 0x30000000.

Example: baseva 0x20000000

- **global-size <n>G|<n>M|<n>** Set the global memory size, memory shared across all router instances, packet buffers, etc. If not set, defaults to 64M. The input value can be set in GB, MB or bytes.

Example: global-size 2G

- **global-pvt-heap-size <n>M|size <n>** Set the size of the global VM private mheap. If not set, defaults to 128k. The input value can be set in MB or bytes.

Example: global-pvt-heap-size size 262144

- **api-pvt-heap-size <n>M|size <n>** Set the size of the api private mheap. If not set, defaults to 128k. The input value can be set in MB or bytes.

Example: api-pvt-heap-size 1M

- **api-size <n>M|<n>G|<n>** Set the size of the API region. If not set, defaults to 16M. The input value can be set in GB, MB or bytes.

Example: api-size 64M

“api-trace” Parameters

The ability to trace, dump, and replay control-plane API traces makes all the difference in the world when trying to understand what the control-plane has tried to ask the forwarding-plane to do.

- **onlenable** Enable API trace capture from the beginning of time, and arrange for a post-mortem dump of the API trace if the application terminates abnormally. By default, the (circular) trace buffer will be configured to capture 256K traces. The default `startup.conf` file has trace enabled by default, and unless there is a very strong reason, it should remain enabled.

Example: on

- **nitems <n>** Configure the circular trace buffer to contain the last <n> entries. By default, the trace buffer captures the last 256K API messages received.

Example: nitems 524288

- **save-api-table <filename>** Dumps the API message table to `/tmp/<filename>`.

Example: save-api-table apiTrace-07-04.txt

Typically, one simply enables the API message trace scheme:

```
api-trace { on }
```

“buffers” Parameters

Command line Buffer configuration controls buffer management.

- **memory-size-in-mb <n>** Configure the memory size used for buffers. If not set, VPP defaults to 32MB.

Example: memory-size-in-mb 64

“cj” Parameters

The circular journal (CJ) thread-safe circular log buffer scheme is occasionally useful when chasing bugs. Calls to it should not be checked in. See `.../vlib/vlib/unix/cj.c`. The circular journal is disabled by default. When enabled, the number of records must be provided, there is no default value.

- **records <n>** Configure the number of circular journal records in the circular buffer. The number of records should be a power of 2.

Example: records 131072

- **on** Turns on logging at the earliest possible moment.

Example: on

“dns” Parameters

- **max-cache-size <n>** TBD

Example: TBD

- **max-ttl <n>** TBD

Example: TBD

“heapsize” Parameters

Heapsize configuration controls the size of the main heap. The heap size is configured very early in the boot sequence, before loading plug-ins or doing much of anything else.

- **heapsize <n>M|<n>G** Specifies the size of the heap in MB or GB. The default is 1GB. Setting the main heap size to 4GB or more requires recompilation of the entire system with `CLIB_VEC64 > 0`. See `.../clib/clib/vec_bootstrap.h`.

Example: heapsize 2G

“ip” Parameters

IPv4 heap configuration. The heap size is configured very early in the boot sequence, before loading plug-ins or doing much of anything else.

- **heap-size <n>G|<n>M|<n>K|<n>** Set the IPv4 mtrie heap size, which is the amount of memory dedicated to the destination IP lookup table. The input value can be set in GB, MB, KB or bytes. The default value is 32MB.

Example: heap-size 64M

“ip6” Parameters

IPv6 heap configuration. The heap size is configured very early in the boot sequence, before loading plug-ins or doing much of anything else.

- **heap-size** **<n>G|<n>M|<n>K|<n>** Set the IPv6 forwarding table heap size. The input value can be set in GB, MB, KB or bytes. The default value is 32MB.

Example: heap-size 64M

- **hash-buckets** **<n>** Set the number of IPv6 forwarding table hash buckets. The default value is 64K (65536).

Example: hash-buckets 131072

“l2learn” Parameters

Configure Layer 2 MAC Address learning parameters.

- **limit** **<n>** Configures the number of L2 (MAC) addresses in the L2 FIB at any one time, which limits the size of the L2 FIB to **<n>** concurrent entries. Defaults to 4M entries (4194304).

Example: limit 8388608

“l2tp” Parameters

IPv6 Layer 2 Tunnelling Protocol Version 3 (IPv6-L2TPv3) configuration controls the method used to locate a specific IPv6-L2TPv3 tunnel. The following settings are mutually exclusive:

- **lookup-v6-src** Lookup tunnel by IPv6 source address.
- **lookup-v6-dst** Lookup tunnel by IPv6 destination address.
- **lookup-session-id** Lookup tunnel by L2TPv3 session identifier.

Example: lookup-v6-src

Example: lookup-v6-dst

Example: lookup-session-id

“logging” Parameters

- **size** **<n>** TBD

Example: TBD

- **unthrottle-time** **<n>** TBD

Example: TBD

- **default-log-level** **emerg|alertcrit|err|warn|notice|info|debug|disabled** TBD

Example: TBD

- **default-syslog-log-level** **emerg|alertcrit|err|warn|notice|info|debug|disabled** TBD

Example: TBD

“mactime” Parameters

- **lookup-table-buckets <n>** TBD

Example: TBD

- **lookup-table-memory <n>G|<n>M|<n>K|<n>**

TBD The input value can be set in GB, MB, KB or bytes. The default value is 256KB.

Example: TBD

- **timezone_offset <n>** TBD

Example: TBD

“map” Parameters

- **customer edge** TBD

Example: customer edge

“mc” Parameters

MC Test Process.

- **interface <name>** TBD

Example: TBD

- **n-bytes <n>** TBD

Example: TBD

- **max-n-bytes <n>** TBD

Example: TBD

- **min-n-bytes <n>** TBD

Example: TBD

- **seed <n>** TBD

Example: TBD

- **window <n>** TBD

Example: TBD

- **verbose** TBD

Example: verbose

- **no-validate** TBD

Example: no-validate

- **min-delay <n.n>** TBD

Example: TBD

- **max-delay <n.n>** TBD

Example: TBD

- **no-delay** TBD

Example: no-delay

- **n-packets <n.n>** TBD

Example: TBD

“nat” Parameters

- **translation hash buckets <n>** TBD

Example: TBD

- **translation hash memory <n>** TBD

Example: TBD

- **user hash buckets <n>** TBD

Example: TBD

- **user hash memory <n>** TBD

Example: TBD

- **max translations per user <n>** TBD

Example: TBD

- **outside VRF id <n>** TBD

Example: TBD

- **outside ip6 VRF id <n>** TBD

Example: TBD

- **inside VRF id <n>** TBD

Example: TBD

- **inside VRF id <n>** TBD

Example: TBD

- **static mapping only** TBD

Example: static mapping only

- **connection tracking** TBD

Example: connection tracking

- **deterministic** TBD

Example: deterministic

- **nat64 bib hash buckets <n>** TBD

Example: TBD

- **nat64 bib hash memory <n>** TBD

Example: TBD

- **nat64 st hash buckets <n>** TBD

Example: TBD

- **nat64 st hash memory <n>** TBD

Example: TBD

- **out2in dpo** TBD

Example: out2in dpo

- **dslite ce** TBD

Example: dslite ce

- **endpoint-dependent** TBD

Example: endpoint-dependent

“oam” Parameters

OAM configuration controls the (ip4-icmp) interval, and number of misses allowed before reporting an oam target down to any registered listener.

- **interval <n.n>** Interval, floating-point seconds, between sending OAM IPv4 ICMP messages. Default is 2.04 seconds.

Example: interval 3.5

- **misses-allowed <n>** Number of misses before declaring an OAM target down. Default is 3 misses.

Example: misses-allowed 5

“plugins” Parameters

A plugin can be disabled by default. It may still be in an experimental phase or only be needed in special circumstances. If this is the case, the plugin can be explicitly enabled in *‘startup.conf’*. Also, a plugin that is enabled by default can be explicitly disabled in *‘startup.conf’*.

Another useful use of this section is to disable all the plugins, then enable only the plugins that are desired.

- **path <path>** Adjust the plugin path depending on where the VPP plugins are installed.

Example: path /home/bms/vpp/build-root/install-vpp-native/vpp/lib64/vpp_plugins

- **name-filter <filter-name>** TBD

Example: TBD

- **vat-path <path>** TBD

Example: TBD

- **vat-name-filter <filter-name>** TBD

Example: TBD

- **plugin <plugin.so> { .. }** Configure parameters for a given plugin. Valid parameters are as follows:

- **enable** Enable the given plugin.
- **disable** Disable the given plugin.
- **skip-version-check** In the plugin registration, if *‘version_required’* is set, the plugin will not be loaded if there is version mismatch between plugin and VPP. This can be bypassed by setting “skip-version-check” for specific plugin.

Example: plugin ila_plugin.so { enable skip-version-check }

- **plugin default { .. }** Set the default behavior for all plugins. Valid parameters are as follows:
 - **disable** Disable all plugins.

Example:

```
plugin default { disable }
plugin dpdk_plugin.so { enable }
plugin acl_plugin.so { enable }
```

“plugin_path” Parameters

Alternate syntax to choose plugin path. Plugin_path configuration controls the set of directories searched for vlib plugins. Supply a colon-separated list of (absolute) directory names: plugin_path dir1:dir2:...:dirN

Example: plugin_path /home/bms/vpp/build-root/install-vpp-native/vpp/lib64/vpp_plugins

“punt” Parameters

- **socket <path>** TBD

Example: TBD

“session” Parameters

- **event-queue-length <n>** TBD

Example: TBD

- **preallocated-sessions <n>** TBD

Example: TBD

- **v4-session-table-buckets <n>** TBD

Example: TBD

- **v4-halfopen-table-buckets <n>** TBD

Example: TBD

- **v6-session-table-buckets <n>** TBD

Example: TBD

- **v6-halfopen-table-buckets <n>** TBD

Example: TBD

- **v4-session-table-memory <n>G|<n>M|<n>K|<n>** TBD The input value can be set in GB, MB, KB or bytes.

Example: TBD

- **v4-halfopen-table-memory <n>G|<n>M|<n>K|<n>** TBD The input value can be set in GB, MB, KB or bytes.

Example: TBD

- **v6-session-table-memory <n>G|<n>M|<n>K|<n>** TBD The input value can be set in GB, MB, KB or bytes.

Example: TBD

- **v6-halfopen-table-memory** <n>G|<n>M|<n>K|<n> TBD The input value can be set in GB, MB, KB or bytes.

Example: TBD

- **local-endpoints-table-memory** <n>G|<n>M|<n>K|<n> TBD The input value can be set in GB, MB, KB or bytes.

Example: TBD

- **local-endpoints-table-buckets** <n> TBD

Example: TBD

- **evt_qs_memfd_seg** TBD

Example: evt_qs_memfd_seg

“socketsvr” Parameters

Create a socket server for API server (.../vlibmemory/socksvr_vlib.c.). If not set, API server doesn't run.

- **socket-name** <filename> Configure API socket filename.

Example: socket-name /run/vpp/vpp-api.sock

- **default** Use the default API socket (/run/vpp-api.sock).

Example: default

“stats” Parameters

Create a socket server for 'stats' poller. If not set, 'stats'* poller doesn't run.

- **socket-name** <filename> Configure 'stats' socket filename.

Example: socket-name /run/vpp/stats.sock

- **default** Use the default 'stats' socket (/run/vpp/stats.sock).

Example: default

“statseg” Parameters

- **size** <n>G|<n>M|<n>K|<n> TBD The input value can be set in GB, MB, KB or bytes.

Example: TBD

“tapcli” Parameters

- **mtu** <n> TBD

Example: TBD

- **disable** TBD

Example: disable

“tcp” Parameters

- **preallocated-connections** <n> TBD

Example: TBD

- **preallocated-half-open-connections** <n> TBD

Example: TBD

- **buffer-fail-fraction** <n.n> TBD

Example: TBD

“tls” Parameters

- **se-test-cert-in-ca** TBD

Example: se-test-cert-in-ca

- **ca-cert-path** <filename> TBD If not set, the default is set to */etc/ssl/certs/ca-certificates.crt*.

Example: TBD

“tuntap” Parameters

The “tuntap” driver configures a point-to-point interface between the vpp engine and the local Linux kernel stack. This allows e.g. users to ssh to the host | VM | container via vpp “revenue” interfaces. It’s marginally useful, and is currently disabled by default. To [dynamically] create TAP interfaces - the preferred scheme - see the “tap_connect” binary API. The Linux network stack “vnet” interface needs to manually configure, and VLAN and other settings if desired.

- **enable/disable** Enable or disable the tun/tap driver.

Example: enable

- **ethernet/ether** Create a tap device (ethernet MAC) instead of a tun device (point-to-point tunnel). The two keywords are aliases for the same function.

Example: ethernet

- **have-normal-interface/have-normal** Treat the host Linux stack as a routing peer instead of programming VPP interface L3 addresses onto the tun/tap devices. The two keywords are aliases for the same function.

Example: have-normal-interface

- **name** <name> Assign name to the tun/tap device.

Example: name vpp1

Here’s a typical multiple parameter invocation:

```
tuntap { ethernet have-normal-interface name vpp1 }
```

“vhost-user” Parameters

Vhost-user configuration parameters control the vhost-user driver.

- **coalesce-frames <n>** Subject to deadline-timer expiration - see next item - attempt to transmit at least <n> packet frames. Default is 32 frames.

Example: coalesce-frames 64

- **coalesce-time <seconds>** Hold packets no longer than (floating-point) seconds before transmitting them. Default is 0.001 seconds

Example: coalesce-time 0.002

- **dont-dump-memory** vhost-user shared-memory segments can add up to a large amount of memory, so it's handy to avoid adding them to corefiles when using a significant number of such interfaces.

Example: dont-dump-memory

“vlib” Parameters

- **memory-trace** TBD

Example: memory-trace

- **elog-events <n>** TBD

Example: TBD

- **elog-post-mortem-dump** TBD

Example: elog-post-mortem-dump

2.1.3 Running VPP

‘vpp’ Usergroup

When VPP is installed, a new usergroup ‘vpp’ is created. To avoid running the VPP CLI (vppctl) as root, add any existing users to the new group that need to interact with VPP:

```
$ sudo usermod -a -G vpp user1
```

Update your current session for the group change to take effect:

```
$ newgrp vpp
```

VPP Systemd File - ‘vpp.service’

When the VPP is installed, a systemd service files is also installed. This file, vpp.service (Ubuntu: /lib/systemd/system/vpp.service and CentOS: /usr/lib/systemd/system/vpp.service), controls how VPP is run as a service. For example, whether or not to restart on failure and if so, with how much delay. Also, which UIO driver should be loaded and location of the ‘startup.conf’ file.

```
$ cat /usr/lib/systemd/system/vpp.service
[Unit]
Description=Vector Packet Processing Process
After=syslog.target network.target auditd.service

[Service]
ExecStartPre=/bin/rm -f /dev/shm/db /dev/shm/global_vm /dev/shm/vpe-api
```

(continues on next page)

(continued from previous page)

```
ExecStartPre=-/sbin/modprobe uio_pci_generic
ExecStart=/usr/bin/vpp -c /etc/vpp/startup.conf
Type=simple
Restart=on-failure
RestartSec=5s

[Install]
WantedBy=multi-user.target
```

Note: Some older versions of the ‘*uio_pci_generic*’ driver don’t bind all the supported NICs properly, so the ‘*igb_uio*’ driver built from DPDK needs to be installed. This file controls which driver is loaded at boot. ‘*startup.conf*’ file controls which driver is used.

2.1.4 VPP Configuration Utility

This guide provides instructions on how to install and use the vpp configuration utility.

The FD.io VPP Configuration Utility, or vpp-config, allows the user to configure FD.io VPP in a simple and safe manner. The utility takes input from the user and creates the configuration files in a dry run directory. The user should then examine these files for correctness. If the configuration files look correct, the user can then apply the configuration. Once the configuration is applied the user should then check the system configuration with the utility and see if it was applied correctly.

This utility also includes a utility that can be used to install or uninstall FD.io VPP packages. This should be used to insure the latest tested release is installed.

Installing VPP-Config

The FD.io VPP configuration utility uses the [Python Package Index](#) or “pypi”.

To install FD.io vpp-config first install python pip.

For Ubuntu execute:

```
$ sudo apt-get install python-pip
```

For Centos execute:

```
$ sudo yum install python-pip
```

Then install the config utility itself.

```
$ sudo -H pip install vpp-config
```

Running

The vpp-config utility provides the user with several different menus. This allows the user to configure ports, some performance characteristics, the number of huge pages and install/uninstall the released FD.io packages.

It is recommended that the menu options are executed in this order.

1. *4) List/Install/Uninstall VPP*

2. *1) Show basic system information*
3. *2) Dry Run*
4. *3) Full Configuration*

Once vpp-config is installed as a root user execute the following

```
$ sudo -H bash
# 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)
4) List/Install/Uninstall VPP.
q) Quit

Command:
```

Default Values

If you do not choose to modify the default for any of the questions prompted by vpp-config, you may press the ENTER key to select the default options:

- Questions that ask [Y/n], the capital letter Y is the default answer.
- Numbers have their default within brackets, such as in [1024], the 1024 is the default.

List/Install/Uninstall VPP

With option “4” the user can list, install or uninstall the FD.io VPP packages. If there are packages already installed, the packages will be listed and then the user will be asked if the packages should be uninstalled. If no packages are installed the user will be asked if the FD.io packages should be installed. The packages installed will be the latest released packages.

Uninstalling the packages:

```
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)
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
q) Quit
```

(continues on next page)

(continued from previous page)

```

Command: 4

These packages are installed on node localhost
Name                               Version
vpp                                18.04-release
vpp-api-java                       18.04-release
vpp-api-lua                       18.04-release
vpp-api-python                    18.04-release
vpp-dbg                           18.04-release
vpp-dev                           18.04-release
vpp-dpdk-dev                      17.01.1-release
vpp-dpdk-dkms                    17.01.1-release
vpp-lib                           18.04-release
vpp-nsh-plugin                   18.04
vpp-nsh-plugin-dbg               18.04
vpp-nsh-plugin-dev               18.04
vpp-plugins                      18.04-release

Do you want to uninstall these packages [y/N]? y
INFO:root: Local Command: service vpp stop
INFO:root:Uninstall Ubuntu
INFO:root: Local Command: dpkg -l | grep vpp
....
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)
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
q) Quit

Command:

```

Installing the packages:

```

1) Show basic system information
2) Dry Run (Will save the configuration files in /usr/local/vpp/vpp-config/dryrun for
  ↳inspection)
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
q) Quit

Command: 4

There are no VPP packages on node localhost.
Do you want to install VPP [Y/n]? Y
INFO:root:  Ubuntu
INFO:root:Install Ubuntu
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
....

What would you like to do?

1) Show basic system information

```

(continues on next page)

(continued from previous page)

```
2) Dry Run (Will save the configuration files in /usr/local/vpp/vpp-config/dryrun for ↵
↵inspection)
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
q) Quit

Command: 4

These packages are installed on node localhost
Name                               Version
vpp                                18.04-release
vpp-api-java                       18.04-release
vpp-api-lua                        18.04-release
vpp-api-python                    18.04-release
vpp-dbg                            18.04-release
vpp-dev                            18.04-release
vpp-dpdk-dev                      17.01.1-release
vpp-dpdk-dkms                     17.01.1-release
vpp-lib                           18.04-release
vpp-nsh-plugin                    18.04
vpp-nsh-plugin-dbg                18.04
vpp-nsh-plugin-dev                18.04
vpp-plugins                       18.04-release

Do you want to uninstall these packages [y/N]? N

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)
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
q) Quit

Command:
```

Show System Information

With option “1” the user can inspect the system characteristics.

This example shows a system that VPP has not yet been installed.

```
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)
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
q) Quit

Command: 1

=====
```

(continues on next page)

(continued from previous page)

```

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=/vmlinuz-4.4.0-128-generic root=UUID=9930aaaa-b1d3-4cf5-b0de-
    ↪d95dbb7dec5e 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      : 65863384 kB
    Total Free Memory        : 45911100 kB
    Actual Huge Page Total   : 1024
    Configured Huge Page Total : 8192
    Huge Pages Free          : 896
    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

No devices bound to DPDK drivers

VPP Service Status:
    Not Installed

```

Dry Run

With the config utility dry run option the important configuration files are created so that the user can examine them and then if they look reasonable apply them with option 3. The files for **Ubuntu** can be found in the root directory

/usr/local/vpp/vpp-config/dryrun and for **Centos** in /usr/vpp/vpp-config/dryrun.

The important configuration files are **/etc/vpp/startup.conf**, **/etc/sysctl.d/80-vpp.conf**, and **/etc/default/grub**

Startup.conf

FD.io VPP startup parameters are configured in the file **/etc/vpp/startup.conf**. The utility creates this file under the vpp-config root directory in vpp/startup.conf. the values in this file come from the questions asked about the devices, cores, rx queues, and tcp parameters.

80-vpp.conf

The huge page configuration comes by setting values in the file **/etc/sysctl.d/80-vpp.conf**. The utility creates the file under the root directory in sysctl.d/80-vpp.conf. When asked the question about huge pages the correct values are put in the dryin file.

grub

CPUs can be isolated for use by VPP or other processes such as VMs using the grub configuration file. This file is **/etc/default/grub**. This file must be modified with care. It is possible to make your system unusable if this file is modified incorrectly. The dry run file is located under the vpp-config root directory and then default.

Executing the Dry Run

The following is an example of how to execute a dry run. Defaults should be picked first and then the values increased accordingly.

```
1) Show basic system information
2) Dry Run (Will save the configuration files in /usr/local/vpp/vpp-config/dryrun for
↪inspection)
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
q) Quit

Command: 2

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

PCI ID           Kernel Interface(s)      Description
-----
↪-----
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
```

(continues on next page)

(continued from previous page)

```

Would you like to use any of these device(s) for VPP [y/N]? y
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.

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
0000:90:00.0	VIC Ethernet NIC

```

Would you like to remove any of these device(s) [y/N]? y
Would you like to remove 0000:86:00.0 [y/N]?
Would you like to remove 0000:86:00.1 [y/N]?
Would you like to remove 0000:90:00.0 [y/N]? y

```

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 start reserving cores and adding queues as needed.

```

How many core(s) shall we reserve for VPP [0-4][0]? 2
How many core(s) do you want to reserve for processes other than VPP? [0-15][0]?
Should we reserve 1 core for the VPP Main thread? [y/N]? y
How many RX queues per port shall we use for VPP [1-4][1]? 2

```

```

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 896 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 - 15644][1024]? 8192

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)
- 3) Full configuration (WARNING: This will change the system configuration)
- 4) List/Install/Uninstall VPP.
- q) Quit

(continues on next page)

(continued from previous page)

Command:

Apply the Configuration

After the configuration files have been examined and are correct. The configuration can be applied. After the configuration is applied use option “1” to check the system configuration. Notice the default is NOT to change the grub file. If the option to change the grub command line is selected a reboot of the system 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)
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
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
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 {
>
```

(continues on next page)

(continued from previous page)

```

> 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]?

```

The Configuration Applied

After the configuration is applied the system parameters should be examined.

```

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)
3) Full configuration (WARNING: This will change the system configuration)
4) List/Install/Uninstall VPP.
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=/vmlinuz-4.4.0-128-generic root=UUID=9930aeee-b1d3-4cf5-b0de-
↪d95dbb7dec5e ro
  Configured: GRUB_CMDLINE_LINUX_DEFAULT="isolcpus=1,8,9-10 nohz_full=1,8,9-10 rcu_
↪nocbs=1,8,9-10"

Huge Pages:
  Total System Memory      : 65863384 kB
  Total Free Memory       : 31169292 kB
  Actual Huge Page Total   : 8196
  Configured Huge Page Total : 8196
  Huge Pages Free         : 7942
  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)

```

CPU

The CPU section of the system information is a summary of the CPU characteristics of the system. It is important to understand the CPU topology and frequency in order to understand what the VPP performance characteristics would be.

Threads

It usually is not needed, but VPP can be configured to run on isolated CPUs. In the example shown VPP is configured with 2 workers. The main thread is also configured to run on a separate CPU. The stats thread will always run on CPU 0. This utility will put the worker threads on CPUs that are associated with the ports that are configured.

Grub Command Line

In general the Grub command line does not need to be changed. If the system is running many processes it may be necessary to isolate CPUs for VPP or other processes.

Huge Pages

As default when VPP is configured the number of huge pages that will be configured will be 1024. This may not be enough. This section will show the total system memory and how many are configured.

Devices

In the devices section we have the “Total Number of Buffers”. This utility allocates the correct number of buffers. The number of buffers are calculated from the number of rx queues.

VPP will not use links that are up. Those devices are shown with this utility.

The devices bound to the kernel are not being used by VPP, but can be.

The devices that are being used by VPP are shown with the interface name be used with VPP. The socket being used by the VPP port is also shown. Notice in this example the worker thread are on the correct CPU. The number of RX, TX Descriptors and TX queues are calculated from the number of RX queues.

VPP Service Status

The VPP service status, will be installed, not installed, running or not.

2.2 Developers

2.2.1 Building VPP

To get started developing with VPP you need to get the sources and build the packages.

Set up Proxies

Depending on the environment, proxies may need to be set. You may run these commands:

```
$ export http_proxy=http://<proxy-server-name>.com:<port-number>
$ export https_proxy=https://<proxy-server-name>.com:<port-number>
```

Get the VPP Sources

To get the VPP sources and get ready to build execute the following:

```
$ git clone https://gerrit.fd.io/r/vpp
$ cd vpp
```

Build VPP Dependencies

Before building, make sure there are no FD.io VPP or DPDK packages installed by entering the following commands:

```
$ dpkg -l | grep vpp
$ dpkg -l | grep DPDK
```

There should be no output, or packages showing after each of the above commands.

Run this to install the dependencies for FD.io VPP. If it hangs during downloading at any point, you may need to set up *proxies for this to work*.

```
$ make install-dep
Hit:1 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Get:2 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]
Get:3 http://security.ubuntu.com/ubuntu xenial-security InRelease [107 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease [107 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [803 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu xenial-updates/main i386 Packages [732 kB]
...
...
Update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jmap to provide /usr/
↳bin/jmap (jmap) in auto mode
Setting up default-jdk-headless (2:1.8-56ubuntu2) ...
Processing triggers for libc-bin (2.23-0ubuntu3) ...
Processing triggers for systemd (229-4ubuntu6) ...
Processing triggers for ureadahead (0.100.0-19) ...
Processing triggers for ca-certificates (20160104ubuntu1) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...

done.
done.
```

Build VPP (Debug Mode)

This build version contains debug symbols which is useful to modify VPP. The command below will build debug version of VPP. This build will come with `/build-root/vpp_debug-native`.

```
$ make build
make[1]: Entering directory '/home/vagrant/vpp-master/build-root'
#### Arch for platform 'vpp' is native ####
#### Finding source for dpdk ####
#### Makefile fragment found in /home/vagrant/vpp-master/build-data/packages/dpdk.mk_
↳####
#### Source found in /home/vagrant/vpp-master/dpdk ####
#### Arch for platform 'vpp' is native ####
```

(continues on next page)

(continued from previous page)

```

@@@ Finding source for vpp @@@
@@@ Makefile fragment found in /home/vagrant/vpp-master/build-data/packages/vpp.mk_
↪@@@
@@@ Source found in /home/vagrant/vpp-master/src @@@
...
...
make[5]: Leaving directory '/home/vagrant/vpp-master/build-root/build-vpp_debug-
↪native/vpp/vpp-api/java'
make[4]: Leaving directory '/home/vagrant/vpp-master/build-root/build-vpp_debug-
↪native/vpp/vpp-api/java'
make[3]: Leaving directory '/home/vagrant/vpp-master/build-root/build-vpp_debug-
↪native/vpp'
make[2]: Leaving directory '/home/vagrant/vpp-master/build-root/build-vpp_debug-
↪native/vpp'
@@@ Installing vpp: nothing to do @@@
make[1]: Leaving directory '/home/vagrant/vpp-master/build-root'

```

Build VPP (Release Version)

To build the release version of FD.io VPP. This build is optimized and will not create debug symbols. This build will come with /build-root/build-vpp-native

```
$ make release
```

Building Necessary Packages

To build the debian packages, one of the following commands below depending on the system:

Building Debian Packages

```
$ make pkg-deb
```

Building RPM Packages

```
$ make pkg-rpm
```

The packages will be found in the build-root directory.

```

$ ls *.deb

If packages built correctly, this should be the Output

vpp_18.07-rc0~456-gb361076_amd64.deb          vpp-dbg_18.07-rc0~456-gb361076_amd64.
↪deb
vpp-api-java_18.07-rc0~456-gb361076_amd64.deb  vpp-dev_18.07-rc0~456-gb361076_amd64.
↪deb
vpp-api-lua_18.07-rc0~456-gb361076_amd64.deb   vpp-lib_18.07-rc0~456-gb361076_amd64.
↪deb
vpp-api-python_18.07-rc0~456-gb361076_amd64.deb vpp-plugins_18.07-rc0~456-gb361076_
↪amd64.deb

```

Packages built installed end up in build-root directory. Finally, the command below installs all built packages.

```
$ sudo bash
# dpkg -i *.deb
```

2.2.2 Software Architecture

The fd.io vpp implementation is a third-generation vector packet processing implementation specifically related to US Patent 7,961,636, as well as earlier work. Note that the Apache-2 license specifically grants non-exclusive patent licenses; we mention this patent as a point of historical interest.

For performance, the vpp dataplane consists of a directed graph of forwarding nodes which process multiple packets per invocation. This schema enables a variety of micro-processor optimizations: pipelining and prefetching to cover dependent read latency, inherent I-cache phase behavior, vector instructions. Aside from hardware input and hardware output nodes, the entire forwarding graph is portable code.

Depending on the scenario at hand, we often spin up multiple worker threads which process ingress-hashes packets from multiple queues using identical forwarding graph replicas.

VPP Layers - Implementation Taxonomy

VNET

VPP networking source

- Devices
- Layer [2, 3, 4]
- Session Management
- Overlays
- Control Plane
- Traffic Management

Plugins

- Plugins can be in-tree:
 - SNAT,
 - Policy ACL,
 - Flow Per Packet,
 - ILA,
 - IOAM,
 - LB,
 - SIXRD,
 - VCGN
- Separate fd.io project:
 - NSH_SFC

VLIB

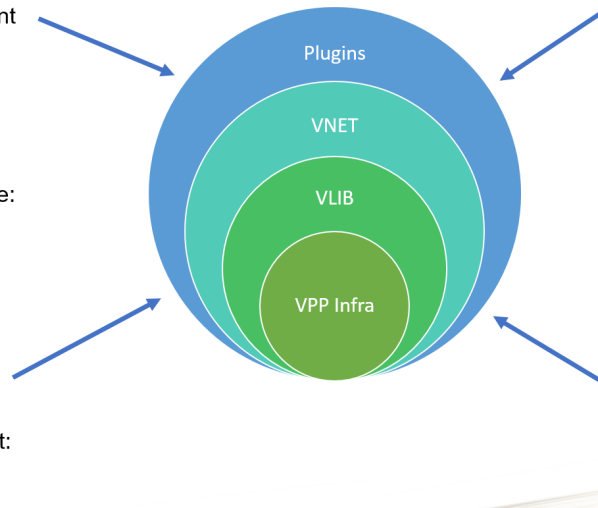
VPP application management

- buffer, buffer management
- graph node, node management
- tracing, counters
- threading
- CLI
- and most importantly ...
- main()

VPP INFRA

Library of function primitives, for

- memory management
- memory operations
- vectors
- rings
- hashing
- timers



- VPP Infra - the VPP infrastructure layer, which contains the core library source code. This layer performs memory functions, works with vectors and rings, performs key lookups in hash tables, and works with timers for dispatching graph nodes.
- VLIB - the vector processing library. The vlib layer also handles various application management functions: buffer, memory and graph node management, maintaining and exporting counters, thread management, packet tracing. Vlib implements the debug CLI (command line interface).
- VNET - works with VPP's networking interface (layers 2, 3, and 4) performs session and traffic management, and works with devices and the data control plane.
- Plugins - Contains an increasingly rich set of data-plane plugins, as noted in the above diagram.

- VPP - the container application linked against all of the above.

It's important to understand each of these layers in a certain amount of detail. Much of the implementation is best dealt with at the API level and otherwise left alone.

2.2.3 VPPINFRA (Infrastructure)

The files associated with the VPP Infrastructure layer are located in the `./src/vppinfra` folder.

Vppinfra is a collection of basic c-library services, quite sufficient to build standalone programs to run directly on bare metal. It also provides high-performance dynamic arrays, hashes, bitmaps, high-precision real-time clock support, fine-grained event-logging, and data structure serialization.

One fair comment / fair warning about vppinfra: you can't always tell a macro from an inline function from an ordinary function simply by name. Macros are used to avoid function calls in the typical case, and to cause (intentional) side-effects.

Vppinfra has been around for almost 20 years and tends not to change frequently. The VPP Infrastructure layer contains the following functions:

Vectors

Vppinfra vectors are ubiquitous dynamically resized arrays with by user defined "headers". Many vppinfra data structures (e.g. hash, heap, pool) are vectors with various different headers.

The memory layout looks like this:

```

User header (optional, uword aligned)
Alignment padding (if needed)
Vector length in elements
User's pointer -> Vector element 0
Vector element 1
...
Vector element N-1
```

As shown above, the vector APIs deal with pointers to the 0th element of a vector. Null pointers are valid vectors of length zero.

To avoid thrashing the memory allocator, one often resets the length of a vector to zero while retaining the memory allocation. Set the vector length field to zero via the `vec_reset_length(v)` macro. [Use the macro! It's smart about NULL pointers.]

Typically, the user header is not present. User headers allow for other data structures to be built atop vppinfra vectors. Users may specify the alignment for data elements via the `vec*_aligned` macros.

Vectors elements can be any C type e.g. (int, double, struct bar). This is also true for data types built atop vectors (e.g. heap, pool, etc.). Many macros have `_a` variants supporting alignment of vector data and `_h` variants supporting non-zero-length vector headers. The `_ha` variants support both.

Inconsistent usage of header and/or alignment related macro variants will cause delayed, confusing failures.

Standard programming error: memorize a pointer to the *i*th element of a vector, and then expand the vector. Vectors expand by 3/2, so such code may appear to work for a period of time. Correct code almost always memorizes vector **indices** which are invariant across reallocations.

In typical application images, one supplies a set of global functions designed to be called from gdb. Here are a few examples:

- `vl(v)` - prints `vec_len(v)`

- `pe(p)` - prints `pool_elts(p)`
- `pifi(p, index)` - prints `pool_is_free_index(p, index)`
- `debug_hex_bytes (p, nbytes)` - hex memory dump `nbytes` starting at `p`

Use the “show gdb” debug CLI command to print the current set.

Bitmaps

Vppinfra bitmaps are dynamic, built using the vppinfra vector APIs. Quite handy for a variety jobs.

Pools

Vppinfra pools combine vectors and bitmaps to rapidly allocate and free fixed-size data structures with independent lifetimes. Pools are perfect for allocating per-session structures.

Hashes

Vppinfra provides several hash flavors. Data plane problems involving packet classification / session lookup often use `./src/vppinfra/bihash_template.[ch]` bounded-index extensible hashes. These templates are instantiated multiple times, to efficiently service different fixed-key sizes.

Bihashes are thread-safe. Read-locking is not required. A simple spin-lock ensures that only one thread writes an entry at a time.

The original vppinfra hash implementation in `./src/vppinfra/hash.[ch]` are simple to use, and are often used in control-plane code which needs exact-string-matching.

In either case, one almost always looks up a key in a hash table to obtain an index in a related vector or pool. The APIs are simple enough, but one must take care when using the unmanaged arbitrary-sized key variant. `Hash_set_mem (hash_table, key_pointer, value)` memorizes `key_pointer`. It is usually a bad mistake to pass the address of a vector element as the second argument to `hash_set_mem`. It is perfectly fine to memorize constant string addresses in the text segment.

Format

Vppinfra format is roughly equivalent to `printf`.

Format has a few properties worth mentioning. Format’s first argument is a `(u8 *)` vector to which it appends the result of the current format operation. Chaining calls is very easy:

```
u8 * result;

result = format (0, "junk = %d, ", junk);
result = format (result, "more junk = %d\n", more_junk);
```

As previously noted, NULL pointers are perfectly proper 0-length vectors. Format returns a `(u8 *)` vector, **not** a C-string. If you wish to print a `(u8 *)` vector, use the “%v” format string. If you need a `(u8 *)` vector which is also a proper C-string, either of these schemes may be used:

```
vec_addl (result, 0)
or
result = format (result, "<whatever>%c", 0);
```

Remember to `vec_free()` the result if appropriate. Be careful not to pass format an uninitialized (`u8 *`).

Format implements a particularly handy user-format scheme via the “%U” format specification. For example:

```
u8 * format_junk (u8 * s, va_list *va)
{
    junk = va_arg (va, u32);
    s = format (s, "%s", junk);
    return s;
}

result = format (0, "junk = %U, format_junk, \"This is some junk\");
```

`format_junk()` can invoke other user-format functions if desired. The programmer shoulders responsibility for argument type-checking. It is typical for user format functions to blow up if the `va_arg(va, type)` macros don’t match the caller’s idea of reality.

Unformat

Vppinfra unformat is vaguely related to `scanf`, but considerably more general.

A typical use case involves initializing an `unformat_input_t` from either a C-string or a (`u8 *`) vector, then parsing via `unformat()` as follows:

```
unformat_input_t input;

unformat_init_string (&input, "<some-C-string>");
/* or */
unformat_init_vector (&input, <u8-vector>);
```

Then loop parsing individual elements:

```
while (unformat_check_input (&input) != UNFORMAT_END_OF_INPUT)
{
    if (unformat (&input, "value1 %d", &value1))
        /* unformat sets value1 */
    else if (unformat (&input, "value2 %d", &value2))
        /* unformat sets value2 */
    else
        return clib_error_return (0, "unknown input '%U'",
                                   format_unformat_error, input);
}
```

As with `format`, `unformat` implements a user-unformat function capability via a “%U” user unformat function scheme.

Vppinfra errors and warnings

Many functions within the vpp dataplane have return-values of type `clib_error_t *`. `Clib_error_t`’s are arbitrary strings with a bit of metadata [fatal, warning] and are easy to announce. Returning a NULL `clib_error_t *` indicates “A-OK, no error.”

`Clib_warning(format-args)` is a handy way to add debugging output; `clib` warnings prepend function:line info to unambiguously locate the message source. `Clib_unix_warning()` adds `perror()`-style Linux system-call information. In production images, `clib_warnings` result in syslog entries.

Serialization

Vppinfra serialization support allows the programmer to easily serialize and unserialize complex data structures.

The underlying primitive serialize/unserialize functions use network byte-order, so there are no structural issues serializing on a little-endian host and unserializing on a big-endian host.

Event-logger, graphical event log viewer

The vppinfra event logger provides very lightweight (sub-100ns) precisely time-stamped event-logging services. See `./src/vppinfra/{elog.c, elog.h}`

Serialization support makes it easy to save and ultimately to combine a set of event logs. In a distributed system running NTP over a local LAN, we find that event logs collected from multiple system elements can be combined with a temporal uncertainty no worse than 50us.

A typical event definition and logging call looks like this:

```
ELOG_TYPE_DECLARE (e) =
{
    .format = "tx-msg: stream %d local seq %d attempt %d",
    .format_args = "i4i4i4",
};
struct { u32 stream_id, local_sequence, retry_count; } * ed;
ed = ELOG_DATA (m->elog_main, e);
ed->stream_id = stream_id;
ed->local_sequence = local_sequence;
ed->retry_count = retry_count;
```

The ELOG_DATA macro returns a pointer to 20 bytes worth of arbitrary event data, to be formatted (offline, not at runtime) as described by format_args. Aside from obvious integer formats, the CLIB event logger provides a couple of interesting additions. The “t4” format pretty-prints enumerated values:

```
ELOG_TYPE_DECLARE (e) =
{
    .format = "get_or_create: %s",
    .format_args = "t4",
    .n_enum_strings = 2,
    .enum_strings = { "old", "new", },
};
```

The “t” format specifier indicates that the corresponding datum is an index in the event’s set of enumerated strings, as shown in the previous event type definition.

The “T” format specifier indicates that the corresponding datum is an index in the event log’s string heap. This allows the programmer to emit arbitrary formatted strings. One often combines this facility with a hash table to keep the event-log string heap from growing arbitrarily large.

Noting the 20-octet limit per-log-entry data field, the event log formatter supports arbitrary combinations of these data types. As in: the “.format” field may contain one or more instances of the following:

- i1 - 8-bit unsigned integer
- i2 - 16-bit unsigned integer
- i4 - 32-bit unsigned integer
- i8 - 64-bit unsigned integer
- f4 - float

- f8 - double
- s - NULL-terminated string - be careful
- sN - N-byte character array
- t1,2,4 - per-event enumeration ID
- T4 - Event-log string table offset

The vpp engine event log is thread-safe, and is shared by all threads. Take care not to serialize the computation. Although the event-logger is about as fast as practicable, it's not appropriate for per-packet use in hard-core data plane code. It's most appropriate for capturing rare events - link up-down events, specific control-plane events and so forth.

The vpp engine has several debug CLI commands for manipulating its event log:

```
vpp# event-logger clear
vpp# event-logger save <filename> # for security, writes into /tmp/<filename>.
                                   # <filename> must not contain '.' or '/'
↪ characters
vpp# show event-logger [all] [<nnn>] # display the event log
                                   # by default, the last 250 entries
```

The event log defaults to 128K entries. The command-line argument "... vlib { elog-events nnn } ..." configures the size of the event log.

As described above, the vpp engine event log is thread-safe and shared. To avoid confusing non-appearance of events logged by worker threads, make sure to code vlib_global_main.elog_main - instead of vm->elog_main. The latter form is correct in the main thread, but will almost certainly produce bad results in worker threads.

G2 graphical event viewer

The g2 graphical event viewer can display serialized vppinfra event logs directly, or via the c2cpel tool.

Todo: please convert wiki page and figures

2.2.4 VLIB (Vector Processing Library)

The files associated with vlib are located in the ./src/{vlib, vlibapi, vlibmemory} folders. These libraries provide vector processing support including graph-node scheduling, reliable multicast support, ultra-lightweight cooperative multi-tasking threads, a CLI, plug in .DLL support, physical memory and Linux epoll support. Parts of this library embody US Patent 7,961,636.

Init function discovery

vlib applications register for various [initialization] events by placing structures and __attribute__((constructor)) functions into the image. At appropriate times, the vlib framework walks constructor-generated singly-linked structure lists, calling the indicated functions. vlib applications create graph nodes, add CLI functions, start cooperative multi-tasking threads, etc. etc. using this mechanism.

vlib applications invariably include a number of VLIB_INIT_FUNCTION (my_init_function) macros.

Each init / configure / etc. function has the return type clib_error_t *. Make sure that the function returns 0 if all is well, otherwise the framework will announce an error and exit.

vlib applications must link against vppinfra, and often link against other libraries such as VNET. In the latter case, it may be necessary to explicitly reference symbol(s) otherwise large portions of the library may be AWOL at runtime.

Node Graph Initialization

vlib packet-processing applications invariably define a set of graph nodes to process packets.

One constructs a `vlib_node_registration_t`, most often via the `VLIB_REGISTER_NODE` macro. At runtime, the framework processes the set of such registrations into a directed graph. It is easy enough to add nodes to the graph at runtime. The framework does not support removing nodes.

vlib provides several types of vector-processing graph nodes, primarily to control framework dispatch behaviors. The type member of the `vlib_node_registration_t` functions as follows:

- `VLIB_NODE_TYPE_PRE_INPUT` - run before all other node types
- `VLIB_NODE_TYPE_INPUT` - run as often as possible, after `pre_input` nodes
- `VLIB_NODE_TYPE_INTERNAL` - only when explicitly made runnable by adding pending frames for processing
- `VLIB_NODE_TYPE_PROCESS` - only when explicitly made runnable. “Process” nodes are actually cooperative multi-tasking threads. They **must** explicitly suspend after a reasonably short period of time.

For a precise understanding of the graph node dispatcher, please read `./src/vlib/main.c:vlib_main_loop`.

Graph node dispatcher

`Vlib_main_loop()` dispatches graph nodes. The basic vector processing algorithm is diabolically simple, but may not be obvious from even a long stare at the code. Here’s how it works: some input node, or set of input nodes, produce a vector of work to process. The graph node dispatcher pushes the work vector through the directed graph, subdividing it as needed, until the original work vector has been completely processed. At that point, the process recurs.

This scheme yields a stable equilibrium in frame size, by construction. Here’s why: as the frame size increases, the per-frame-element processing time decreases. There are several related forces at work; the simplest to describe is the effect of vector processing on the CPU L1 I-cache. The first frame element [packet] processed by a given node warms up the node dispatch function in the L1 I-cache. All subsequent frame elements profit. As we increase the number of frame elements, the cost per element goes down.

Under light load, it is a crazy waste of CPU cycles to run the graph node dispatcher flat-out. So, the graph node dispatcher arranges to wait for work by sitting in a timed `epoll` wait if the prevailing frame size is low. The scheme has a certain amount of hysteresis to avoid constantly toggling back and forth between interrupt and polling mode. Although the graph dispatcher supports interrupt and polling modes, our current default device drivers do not.

The graph node scheduler uses a hierarchical timer wheel to reschedule process nodes upon timer expiration.

Graph dispatcher internals

This section may be safely skipped. It’s not necessary to understand graph dispatcher internals to create graph nodes.

Vector Data Structure

In `vpp / vlib`, we represent vectors as instances of the `vlib_frame_t` type:

```
typedef struct vlib_frame_t
{
    /* Frame flags. */
    u16 flags;

    /* Number of scalar bytes in arguments. */
```

(continues on next page)

(continued from previous page)

```

u8 scalar_size;

/* Number of bytes per vector argument. */
u8 vector_size;

/* Number of vector elements currently in frame. */
u16 n_vectors;

/* Scalar and vector arguments to next node. */
u8 arguments[0];
} vlib_frame_t;

```

Note that one *could* construct all kinds of vectors - including vectors with some associated scalar data - using this structure. In the vpp application, vectors typically use a 4-byte vector element size, and zero bytes' worth of associated per-frame scalar data.

Frames are always allocated on CLIB_CACHE_LINE_BYTES boundaries. Frames have u32 indices which make use of the alignment property, so the maximum feasible main heap offset of a frame is CLIB_CACHE_LINE_BYTES * 0xFFFFFFFF: 64*4 = 256 Gbytes.

Scheduling Vectors

As you can see, vectors are not directly associated with graph nodes. We represent that association in a couple of ways. The simplest is the vlib_pending_frame_t:

```

/* A frame pending dispatch by main loop. */
typedef struct
{
    /* Node and runtime for this frame. */
    u32 node_runtime_index;

    /* Frame index (in the heap). */
    u32 frame_index;

    /* Start of next frames for this node. */
    u32 next_frame_index;

    /* Special value for next_frame_index when there is no next frame. */
#define VLIB_PENDING_FRAME_NO_NEXT_FRAME ((u32) ~0)
} vlib_pending_frame_t;

```

Here is the code in .../src/vlib/main.c:vlib_main_or_worker_loop() which processes frames:

```

/*
 * Input nodes may have added work to the pending vector.
 * Process pending vector until there is nothing left.
 * All pending vectors will be processed from input -> output.
 */
for (i = 0; i < _vec_len (nm->pending_frames); i++)
    cpu_time_now = dispatch_pending_node (vm, i, cpu_time_now);
/* Reset pending vector for next iteration. */

```

The pending frame node_runtime_index associates the frame with the node which will process it.

Complications

Fasten your seatbelt. Here's where the story - and the data structures - become quite complicated...

At 100,000 feet: vpp uses a directed graph, not a directed *acyclic* graph. It's really quite normal for a packet to visit ip[46]-lookup multiple times. The worst-case: a graph node which enqueues packets to itself.

To deal with this issue, the graph dispatcher must force allocation of a new frame if the current graph node's dispatch function happens to enqueue a packet back to itself.

There are no guarantees that a pending frame will be processed immediately, which means that more packets may be added to the underlying vlib_frame_t after it has been attached to a vlib_pending_frame_t. Care must be taken to allocate new frames and pending frames if a (pending_frame, frame) pair fills.

Next frames, next frame ownership

The vlib_next_frame_t is the last key graph dispatcher data structure:

```
typedef struct
{
    /* Frame index. */
    u32 frame_index;

    /* Node runtime for this next. */
    u32 node_runtime_index;

    /* Next frame flags. */
    u32 flags;

    /* Reflects node frame-used flag for this next. */
#define VLIB_FRAME_NO_FREE_AFTER_DISPATCH \
    VLIB_NODE_FLAG_FRAME_NO_FREE_AFTER_DISPATCH

    /* This next frame owns enqueue to node
       corresponding to node_runtime_index. */
#define VLIB_FRAME_OWNER (1 << 15)

    /* Set when frame has been allocated for this next. */
#define VLIB_FRAME_IS_ALLOCATED VLIB_NODE_FLAG_IS_OUTPUT

    /* Set when frame has been added to pending vector. */
#define VLIB_FRAME_PENDING VLIB_NODE_FLAG_IS_DROP

    /* Set when frame is to be freed after dispatch. */
#define VLIB_FRAME_FREE_AFTER_DISPATCH VLIB_NODE_FLAG_IS_PUNT

    /* Set when frame has traced packets. */
#define VLIB_FRAME_TRACE VLIB_NODE_FLAG_TRACE

    /* Number of vectors enqueue to this next since last overflow. */
    u32 vectors_since_last_overflow;
} vlib_next_frame_t;
```

Graph node dispatch functions call vlib_get_next_frame (...) to set "(u32 *)to_next" to the right place in the vlib_frame_t corresponding to the ith arc (aka next0) from the current node to the indicated next node.

After some scuffling around - two levels of macros - processing reaches vlib_get_next_frame_internal (...). Get-next-frame-internal digs up the vlib_next_frame_t corresponding to the desired graph arc.

The next frame data structure amounts to a graph-arc-centric frame cache. Once a node finishes adding element to a frame, it will acquire a `vlib_pending_frame_t` and end up on the graph dispatcher's run-queue. But there's no guarantee that more vector elements won't be added to the underlying frame from the same (`source_node`, `next_index`) arc or from a different (`source_node`, `next_index`) arc.

Maintaining consistency of the arc-to-frame cache is necessary. The first step in maintaining consistency is to make sure that only one graph node at a time thinks it "owns" the target `vlib_frame_t`.

Back to the graph node dispatch function. In the usual case, a certain number of packets will be added to the `vlib_frame_t` acquired by calling `vlib_get_next_frame(...)`.

Before a dispatch function returns, it's required to call `vlib_put_next_frame(...)` for all of the graph arcs it actually used. This action adds a `vlib_pending_frame_t` to the graph dispatcher's pending frame vector.

`Vlib_put_next_frame` makes a note in the pending frame of the frame index, and also of the `vlib_next_frame_t` index.

dispatch_pending_node actions

The main graph dispatch loop calls `dispatch_pending_node` as shown above.

`Dispatch_pending_node` recovers the pending frame, and the graph node runtime / dispatch function. Further, it recovers the `next_frame` currently associated with the `vlib_frame_t`, and detaches the `vlib_frame_t` from the `next_frame`.

In `.../src/vlib/main.c:dispatch_pending_node(...)`, note this stanza:

```
/* Force allocation of new frame while current frame is being
   dispatched. */
restore_frame_index = ~0;
if (nf->frame_index == p->frame_index)
{
    nf->frame_index = ~0;
    nf->flags &= ~VLIB_FRAME_IS_ALLOCATED;
    if (!(n->flags & VLIB_NODE_FLAG_FRAME_NO_FREE_AFTER_DISPATCH))
        restore_frame_index = p->frame_index;
}
```

`dispatch_pending_node` is worth a hard stare due to the several second-order optimizations it implements. Almost as an afterthought, it calls `dispatch_node` which actually calls the graph node dispatch function.

Process / thread model

`vlib` provides an ultra-lightweight cooperative multi-tasking thread model. The graph node scheduler invokes these processes in much the same way as traditional vector-processing run-to-completion graph nodes; plus-or-minus a `setjmp/longjmp` pair required to switch stacks. Simply set the `vlib_node_registration_t` type field to `vlib_NODE_TYPE_PROCESS`. Yes, process is a misnomer. These are cooperative multi-tasking threads.

As of this writing, the default stack size is $2^{15} = 32\text{kb}$. Initialize the node registration's `process_log2_n_stack_bytes` member as needed. The graph node dispatcher makes some effort to detect stack overrun, e.g. by mapping a no-access page below each thread stack.

Process node dispatch functions are expected to be "while(1) { }" loops which suspend when not otherwise occupied, and which must not run for unreasonably long periods of time.

"Unreasonably long" is an application-dependent concept. Over the years, we have constructed frame-size sensitive control-plane nodes which will use a much higher fraction of the available CPU bandwidth when the frame size is low. The classic example: modifying forwarding tables. So long as the table-builder leaves the forwarding tables in a valid state, one can suspend the table builder to avoid dropping packets as a result of control-plane activity.

Process nodes can suspend for fixed amounts of time, or until another entity signals an event, or both. See the next section for a description of the vlib process event mechanism.

When running in vlib process context, one must pay strict attention to loop invariant issues. If one walks a data structure and calls a function which may suspend, one had best know by construction that it cannot change. Often, it's best to simply make a snapshot copy of a data structure, walk the copy at leisure, then free the copy.

Process events

The vlib process event mechanism API is extremely lightweight and easy to use. Here is a typical example:

```
vlib_main_t *vm = &vlib_global_main;
uword event_type, * event_data = 0;

while (1)
{
    vlib_process_wait_for_event_or_clock (vm, 5.0 /* seconds */);

    event_type = vlib_process_get_events (vm, &event_data);

    switch (event_type) {
    case EVENT1:
        handle_event1s (event_data);
        break;

    case EVENT2:
        handle_event2s (event_data);
        break;

    case ~0: /* 5-second idle/periodic */
        handle_idle ();
        break;

    default: /* bug! */
        ASSERT (0);
    }

    vec_reset_length(event_data);
}
```

In this example, the VLIB process node waits for an event to occur, or for 5 seconds to elapse. The code demuxes on the event type, calling the appropriate handler function. Each call to `vlib_process_get_events` returns a vector of per-event-type data passed to successive `vlib_process_signal_event` calls; it is a serious error to process only `event_data[0]`.

Resetting the `event_data` vector-length to 0 [instead of calling `vec_free`] means that the event scheme doesn't burn cycles continuously allocating and freeing the event data vector. This is a common vppinfra / vlib coding pattern, well worth using when appropriate.

Signaling an event is easy, for example:

```
vlib_process_signal_event (vm, process_node_index, EVENT1,
    (uword)arbitrary_event1_data); /* and so forth */
```

One can either know the process node index by construction - dig it out of the appropriate `vlib_node_registration_t` - or by finding the `vlib_node_t` with `vlib_get_node_by_name(...)`.

Buffers

vlib buffering solves the usual set of packet-processing problems, albeit at high performance. Key in terms of performance: one ordinarily allocates / frees N buffers at a time rather than one at a time. Except when operating directly on a specific buffer, one deals with buffers by index, not by pointer.

Packet-processing frames are u32[] arrays, not vlib_buffer_t[] arrays.

Packets comprise one or more vlib buffers, chained together as required. Multiple packet sizes are supported; hardware input nodes simply ask for the required size(s). Coalescing support is available. For obvious reasons one is discouraged from writing one's own wild and wacky buffer chain traversal code.

vlib buffer headers are allocated immediately prior to the buffer data area. In typical packet processing this saves a dependent read wait: given a buffer's address, one can prefetch the buffer header [metadata] at the same time as the first cache line of buffer data.

Buffer header metadata (vlib_buffer_t) includes the usual rewrite expansion space, a current_data offset, RX and TX interface indices, packet trace information, and a opaque areas.

The opaque data is intended to control packet processing in arbitrary subgraph-dependent ways. The programmer shoulders responsibility for data lifetime analysis, type-checking, etc.

Buffers have reference-counts in support of e.g. multicast replication.

Shared-memory message API

Local control-plane and application processes interact with the vpp dataplane via asynchronous message-passing in shared memory over unidirectional queues. The same application APIs are available via sockets.

Capturing API traces and replaying them in a simulation environment requires a disciplined approach to the problem. This seems like a make-work task, but it is not. When something goes wrong in the control-plane after 300,000 or 3,000,000 operations, high-speed replay of the events leading up to the accident is a huge win.

The shared-memory message API message allocator vl_api_msg_alloc uses a particularly cute trick. Since messages are processed in order, we try to allocate message buffering from a set of fixed-size, preallocated rings. Each ring item has a "busy" bit. Freeing one of the preallocated message buffers merely requires the message consumer to clear the busy bit. No locking required.

Debug CLI

Adding debug CLI commands to VLIB applications is very simple.

Here is a complete example:

```
static clib_error_t *
show_ip_tuple_match (vlib_main_t * vm,
                    unformat_input_t * input,
                    vlib_cli_command_t * cmd)
{
    vlib_cli_output (vm, "%U\n", format_ip_tuple_match_tables, &routing_main);
    return 0;
}

/* *INDENT-OFF* */
static VLIB_CLI_COMMAND (show_ip_tuple_command) =
{
    .path = "show ip tuple match",
    .short_help = "Show ip 5-tuple match-and-broadcast tables",

```

(continues on next page)

(continued from previous page)

```
.function = show_ip_tuple_match,  
};  
/* *INDENT-ON* */
```

This example implements the “show ip tuple match” debug cli command. In ordinary usage, the vlib cli is available via the “vppctl” application, which sends traffic to a named pipe. One can configure debug CLI telnet access on a configurable port.

The cli implementation has an output redirection facility which makes it simple to deliver cli output via shared-memory API messaging,

Particularly for debug or “show tech support” type commands, it would be wasteful to write vlib application code to pack binary data, write more code elsewhere to unpack the data and finally print the answer. If a certain cli command has the potential to hurt packet processing performance by running for too long, do the work incrementally in a process node. The client can wait.

2.2.5 Plugins

vlib implements a straightforward plug-in DLL mechanism. VLIB client applications specify a directory to search for plug-in .DLLs, and a name filter to apply (if desired). VLIB needs to load plug-ins very early.

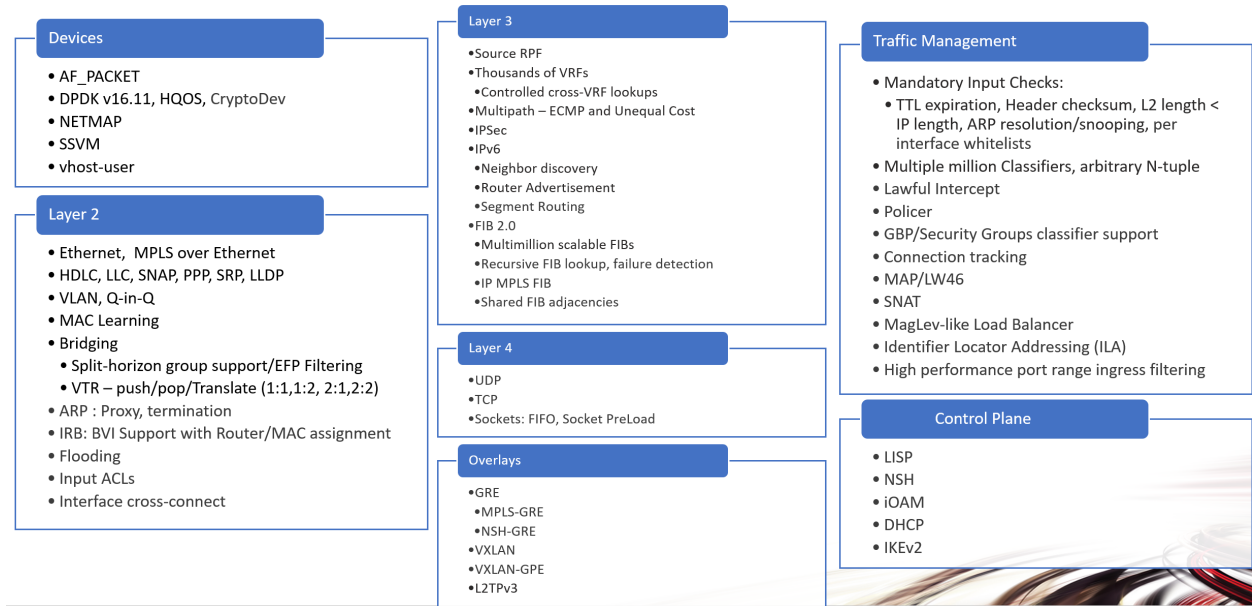
Once loaded, the plug-in DLL mechanism uses dlsym to find and verify a vlib_plugin_registration data structure in the newly-loaded plug-in.

2.2.6 VNET (VPP Network Stack)

The files associated with the VPP network stack layer are located in the ./src/vnet folder. The Network Stack Layer is basically an instantiation of the code in the other layers. This layer has a vnet library that provides vectorized layer-2 and 3 networking graph nodes, a packet generator, and a packet tracer.

In terms of building a packet processing application, vnet provides a platform-independent subgraph to which one connects a couple of device-driver nodes.

Typical RX connections include “ethernet-input” [full software classification, feeds ipv4-input, ipv6-input, arp-input etc.] and “ipv4-input-no-checksum” [if hardware can classify, perform ipv4 header checksum].



List of features and layer areas that VNET works with:

Effective graph dispatch function coding

Over the 15 years, multiple coding styles have emerged: a single/dual/quad loop coding model (with variations) and a fully-pipelined coding model.

Single/dual loops

The single/dual/quad loop model variations conveniently solve problems where the number of items to process is not known in advance: typical hardware RX-ring processing. This coding style is also very effective when a given node will not need to cover a complex set of dependent reads.

Here is an quad/single loop which can leverage up-to-avx512 SIMD vector units to convert buffer indices to buffer pointers:

```
static uword
simulated_ethernet_interface_tx (vlib_main_t * vm,
                                vlib_node_runtime_t *
                                node, vlib_frame_t * frame)
{
    u32 n_left_from, *from;
    u32 next_index = 0;
    u32 n_bytes;
    u32 thread_index = vm->thread_index;
    vnet_main_t *vnm = vnet_get_main ();
    vnet_interface_main_t *im = &vnm->interface_main;
    vlib_buffer_t *bufs[VLIB_FRAME_SIZE], **b;
    u16 nexts[VLIB_FRAME_SIZE], *next;

    n_left_from = frame->n_vectors;
    from = vlib_frame_args (frame);

    /*
```

(continues on next page)

(continued from previous page)

```

    * Convert up to VLIB_FRAME_SIZE indices in "from" to
    * buffer pointers in bufs[]
    */
vlib_get_buffers (vm, from, bufs, n_left_from);
b = bufs;
next = nexts;

/*
 * While we have at least 4 vector elements (pkts) to process..
 */
while (n_left_from >= 4)
{
    /* Prefetch next quad-loop iteration. */
    if (PREDICT_TRUE (n_left_from >= 8))
    {
        vlib_prefetch_buffer_header (b[4], STORE);
        vlib_prefetch_buffer_header (b[5], STORE);
        vlib_prefetch_buffer_header (b[6], STORE);
        vlib_prefetch_buffer_header (b[7], STORE);
    }

    /*
     * $$$ Process 4x packets right here...
     * set next[0..3] to send the packets where they need to go
     */

    do_something_to (b[0]);
    do_something_to (b[1]);
    do_something_to (b[2]);
    do_something_to (b[3]);

    /* Process the next 0..4 packets */
    b += 4;
    next += 4;
    n_left_from -= 4;
}

/*
 * Clean up 0...3 remaining packets at the end of the incoming frame
 */
while (n_left_from > 0)
{
    /*
     * $$$ Process one packet right here...
     * set next[0..3] to send the packets where they need to go
     */
    do_something_to (b[0]);

    /* Process the next packet */
    b += 1;
    next += 1;
    n_left_from -= 1;
}

/*
 * Send the packets along their respective next-node graph arcs
 * Considerable locality of reference is expected, most if not all
 * packets in the inbound vector will traverse the same next-node

```

(continues on next page)

(continued from previous page)

```

    * arc
    */
    vlib_buffer_enqueue_to_next (vm, node, from, nexts, frame->n_vectors);

    return frame->n_vectors;
}

```

Given a packet processing task to implement, it pays to scout around looking for similar tasks, and think about using the same coding pattern. It is not uncommon to recode a given graph node dispatch function several times during performance optimization.

Packet tracer

Vlib includes a frame element [packet] trace facility, with a simple vlib cli interface. The cli is straightforward: “trace add input-node-name count”.

To trace 100 packets on a typical x86_64 system running the dpdk plugin: “trace add dpdk-input 100”. When using the packet generator: “trace add pg-input 100”

Each graph node has the opportunity to capture its own trace data. It is almost always a good idea to do so. The trace capture APIs are simple.

The packet capture APIs snapshot binary data, to minimize processing at capture time. Each participating graph node initialization provides a vppinfra format-style user function to pretty-print data when required by the VLIB “show trace” command.

Set the VLIB node registration “.format_trace” member to the name of the per-graph node format function.

Here’s a simple example:

```

u8 * my_node_format_trace (u8 * s, va_list * args)
{
    vlib_main_t * vm = va_arg (*args, vlib_main_t *);
    vlib_node_t * node = va_arg (*args, vlib_node_t *);
    my_node_trace_t * t = va_arg (*args, my_node_trace_t *);

    s = format (s, "My trace data was: %d", t-><whatever>);

    return s;
}

```

The trace framework hands the per-node format function the data it captured as the packet whizzed by. The format function pretty-prints the data as desired.

2.2.7 Feature Arcs

A significant number of vpp features are configurable on a per-interface or per-system basis. Rather than ask feature coders to manually construct the required graph arcs, we built a general mechanism to manage these mechanics.

Specifically, feature arcs comprise ordered sets of graph nodes. Each feature node in an arc is independently controlled. Feature arc nodes are generally unaware of each other. Handing a packet to “the next feature node” is quite inexpensive.

The feature arc implementation solves the problem of creating graph arcs used for steering.

At the beginning of a feature arc, a bit of setup work is needed, but only if at least one feature is enabled on the arc.

On a per-arc basis, individual feature definitions create a set of ordering dependencies. Feature infrastructure performs a topological sort of the ordering dependencies, to determine the actual feature order. Missing dependencies **will** lead to runtime disorder. See <https://gerrit.fd.io/r/#/c/12753> for an example.

If no partial order exists, vpp will refuse to run. Circular dependency loops of the form “a then b, b then c, c then a” are impossible to satisfy.

Adding a feature to an existing feature arc

To nobody’s great surprise, we set up feature arcs using the typical “macro -> constructor function -> list of declarations” pattern:

```
VNET_FEATURE_INIT (mactime, static) =
{
    .arc_name = "device-input",
    .node_name = "mactime",
    .runs_before = VNET_FEATURES ("ethernet-input"),
};
```

This creates a “mactime” feature on the “device-input” arc.

Once per frame, dig up the `vnet_feature_config_main_t` corresponding to the “device-input” feature arc:

```
vnet_main_t *vnm = vnet_get_main ();
vnet_interface_main_t *im = &vnm->interface_main;
u8 arc = im->output_feature_arc_index;
vnet_feature_config_main_t *fcm;

fcm = vnet_feature_get_config_main (arc);
```

Note that in this case, we’ve stored the required arc index - assigned by the feature infrastructure - in the `vnet_interface_main_t`. Where to put the arc index is a programmer’s decision when creating a feature arc.

Per packet, set `next0` to steer packets to the next node they should visit:

```
vnet_get_config_data (&fcm->config_main,
    &b0->current_config_index /* value-result */,
    &next0, 0 /* # bytes of config data */);
```

Configuration data is per-feature arc, and is often unused. Note that it’s normal to reset `next0` to divert packets elsewhere; often, to drop them for cause:

```
next0 = MACTIME_NEXT_DROP;
b0->error = node->errors[DROP_CAUSE];
```

Creating a feature arc

Once again, we create feature arcs using constructor macros:

```
VNET_FEATURE_ARC_INIT (ip4_unicast, static) =
{
    .arc_name = "ip4-unicast",
    .start_nodes = VNET_FEATURES ("ip4-input", "ip4-input-no-checksum"),
    .arc_index_ptr = &ip4_main.lookup_main.ucast_feature_arc_index,
};
```

In this case, we configure two arc start nodes to handle the “hardware-verified ip checksum or not” cases. During initialization, the feature infrastructure stores the arc index as shown.

In the head-of-arc node, do the following to send packets along the feature arc:

```
ip_lookup_main_t *lm = &im->lookup_main;
arc = lm->ucast_feature_arc_index;
```

Once per packet, initialize packet metadata to walk the feature arc:

```
vnet_feature_arc_start (arc, sw_if_index0, &next, b0);
```

Enabling / Disabling features

Simply call `vnet_feature_enable_disable` to enable or disable a specific feature:

```
vnet_feature_enable_disable ("device-input", /* arc name */
                           "mactime",      /* feature name */
                           sw_if_index,    /* Interface sw_if_index */
                           enable_disable, /* 1 => enable */
                           0 /* (void *) feature_configuration */,
                           0 /* feature_configuration_nbytes */);
```

The `feature_configuration` opaque is seldom used.

If you wish to make a feature a *de facto* system-level concept, pass `sw_if_index=0` at all times. `Sw_if_index 0` is always valid, and corresponds to the “local” interface.

Related “show” commands

To display the entire set of features, use “show features [verbose]”. The verbose form displays arc indices, and feature indicies within the arcs

```
$ vppctl show features verbose
Available feature paths
<snip>
[14] ip4-unicast:
  [ 0]: nat64-out2in-handoff
  [ 1]: nat64-out2in
  [ 2]: nat44-ed-hairpin-dst
  [ 3]: nat44-hairpin-dst
  [ 4]: ip4-dhcp-client-detect
  [ 5]: nat44-out2in-fast
  [ 6]: nat44-in2out-fast
  [ 7]: nat44-handoff-classify
  [ 8]: nat44-out2in-worker-handoff
  [ 9]: nat44-in2out-worker-handoff
[10]: nat44-ed-classify
[11]: nat44-ed-out2in
[12]: nat44-ed-in2out
[13]: nat44-det-classify
[14]: nat44-det-out2in
[15]: nat44-det-in2out
[16]: nat44-classify
[17]: nat44-out2in
[18]: nat44-in2out
```

(continues on next page)

(continued from previous page)

```

[19]: ip4-qos-record
[20]: ip4-vxlan-gpe-bypass
[21]: ip4-reassembly-feature
[22]: ip4-not-enabled
[23]: ip4-source-and-port-range-check-rx
[24]: ip4-flow-classify
[25]: ip4-inacl
[26]: ip4-source-check-via-rx
[27]: ip4-source-check-via-any
[28]: ip4-policer-classify
[29]: ipsec-input-ip4
[30]: vpath-input-ip4
[31]: ip4-vxlan-bypass
[32]: ip4-lookup
<snip>

```

Here, we learn that the ip4-unicast feature arc has index 14, and that e.g. ip4-inacl is the 25th feature in the generated partial order.

To display the features currently active on a specific interface, use “show interface features”:

```

$ vppctl show interface GigabitEthernet3/0/0 features
Feature paths configured on GigabitEthernet3/0/0...
<snip>
ip4-unicast:
    nat44-out2in
<snip>

```

Table of Feature Arcs

Simply search for name-strings to track down the arc definition, location of the arc index, etc.

Arc Name
device-input
ethernet-output
interface-output
ip4-drop
ip4-local
ip4-multicast
ip4-output
ip4-punt
ip4-unicast
ip6-drop
ip6-local
ip6-multicast
ip6-output
ip6-punt
ip6-unicast
mpls-input
mpls-output
nsh-output

2.2.8 Bounded-index Extensible Hashing (bihash)

Vpp uses bounded-index extensible hashing to solve a variety of exact-match (key, value) lookup problems. Benefits of the current implementation:

- Very high record count scaling, tested to 100,000,000 records.
- Lookup performance degrades gracefully as the number of records increases
- No reader locking required
- Template implementation, it's easy to support arbitrary (key,value) types

Bounded-index extensible hashing has been widely used in databases for decades.

Bihash uses a two-level data structure:



Discussion of the algorithm

This structure has a couple of major advantages. In practice, each bucket entry fits into a 64-bit integer. Coincidentally, vpp's target CPU architectures support 64-bit atomic operations. When modifying the contents of a specific bucket, we do the following:

- Make a working copy of the bucket's backing storage
- Atomically swap a pointer to the working copy into the bucket array
- Change the original backing store data
- Atomically swap back to the original

So, no reader locking is required to search a bihash table.

At lookup time, the implementation computes a key hash code. We use the least-significant N bits of the hash to select the bucket.

With the bucket in hand, we learn $\log_2(\text{nBackingPages})$ for the selected bucket. At this point, we use the next \log_2_size bits from the hash code to select the specific backing page in which the (key,value) page will be found.

Net result: we search **one** backing page, not $2^{\log_2_size}$ pages. This is a key property of the algorithm.

When sufficient collisions occur to fill the backing pages for a given bucket, we double the bucket size, rehash, and deal the bucket contents into a double-sized set of backing pages. In the future, we may represent the size as a linear combination of two powers-of-two, to increase space efficiency.

To solve the “jackpot case” where a set of records collide under hashing in a bad way, the implementation will fall back to linear search across $2^{\log_2 \text{size}}$ backing pages on a per-bucket basis.

To maintain *space* efficiency, we should configure the bucket array so that backing pages are effectively utilized. Lookup performance tends to change *very little* if the bucket array is too small or too large.

Bihash depends on selecting an effective hash function. If one were to use a truly broken hash function such as “return 1ULL.” bihash would still work, but it would be equivalent to poorly-programmed linear search.

We often use cpu intrinsic functions - think `crc32` - to rapidly compute a hash code which has decent statistics.

Bihash Cookbook

Using current (key,value) template instance types

It’s quite easy to use one of the template instance types. As of this writing, `.../src/vppinfra` provides pre-built templates for 8, 16, 20, 24, 40, and 48 byte keys, `u8 * vector` keys, and 8 byte values.

See `.../src/vppinfra/{bihash_<key-size>_8}.h`

To define the data types, `#include` a specific template instance, most often in a subsystem header file:

```
#include <vppinfra/bihash_8_8.h>
```

If you’re building a standalone application, you’ll need to define the various functions by `#including` the method implementation file in a C source file.

The core vpp engine currently uses most if not all of the known bihash types, so you probably won’t need to `#include` the method implementation file.

```
#include <vppinfra/bihash_template.c>
```

Add an instance of the selected bihash data structure to e.g. a “`main_t`” structure:

```
typedef struct
{
    ...
    BVT (clib_bihash) hash;
    or
    clib_bihash_8_8_t hash;
    ...
} my_main_t;
```

The `BV` macro concatenate its argument with the value of the preprocessor symbol `BIHASH_TYPE`. The `BVT` macro concatenates its argument with the value of `BIHASH_TYPE` and the fixed-string “`_t`”. So in the above example, `BVT (clib_bihash)` generates “`clib_bihash_8_8_t`”.

If you’re sure you won’t decide to change the template / type name later, it’s perfectly OK to code “`clib_bihash_8_8_t`” and so forth.

In fact, if you `#include` multiple template instances in a single source file, you **must** use fully-enumerated type names. The macros stand no chance of working.

Initializing a bihash table

Call the init function as shown. As a rough guide, pick a number of buckets which is approximately `number_of_expected_records/BIHASH_KVP_PER_PAGE` from the relevant template instance header-file. See previous discussion.

The amount of memory selected should easily contain all of the records, with a generous allowance for hash collisions. Bihash memory is allocated separately from the main heap, and won't cost anything except kernel PTE's until touched, so it's OK to be reasonably generous.

For example:

```
my_main_t *mm = &my_main;
clib_bihash_8_8_t *h;

h = &mm->hash_table;

clib_bihash_init_8_8 (h, "test", (u32) number_of_buckets,
                     (uword) memory_size);
```

Add or delete a key/value pair

Use `BV(clib_bihash_add_del)`, or the explicit type variant:

```
clib_bihash_kv_8_8_t kv;
clib_bihash_8_8_t * h;
my_main_t *mm = &my_main;
clib_bihash_8_8_t *h;

h = &mm->hash_table;
kv.key = key_to_add_or_delete;
kv.value = value_to_add_or_delete;

clib_bihash_add_del_8_8 (h, &kv, is_add /* 1=add, 0=delete */);
```

In the delete case, `kv.value` is irrelevant. To change the value associated with an existing (key,value) pair, simply re-add the [new] pair.

Simple search

The simplest possible (key, value) search goes like so:

```
clib_bihash_kv_8_8_t search_kv, return_kv;
clib_bihash_8_8_t * h;
my_main_t *mm = &my_main;
clib_bihash_8_8_t *h;

h = &mm->hash_table;
search_kv.key = key_to_add_or_delete;

if (clib_bihash_search_8_8 (h, &search_kv, &return_kv) < 0)
    key_not_found()
else
    key_not_found();
```

Note that it's perfectly fine to collect the lookup result

```
if (clib_bihash_search_8_8 (h, &search_kv, &search_kv))
    key_not_found();
etc.
```

Bihash vector processing

When processing a vector of packets which need a certain lookup performed, it's worth the trouble to compute the key hash, and prefetch the correct bucket ahead of time.

Here's a sketch of one way to write the required code:

Dual-loop:

- 6 packets ahead, prefetch 2x vlib_buffer_t's and 2x packet data required to form the record keys
- 4 packets ahead, form 2x record keys and call BV(clib_bihash_hash) or the explicit hash function to calculate the record hashes. Call 2x BV(clib_bihash_prefetch_bucket) to prefetch the buckets
- 2 packets ahead, call 2x BV(clib_bihash_prefetch_data) to prefetch 2x (key,value) data pages.
- In the processing section, call 2x BV(clib_bihash_search_inline_with_hash) to perform the search

Programmer's choice whether to stash the hash code somewhere in vnet_buffer(b) metadata, or to use local variables.

Single-loop:

- Use simple search as shown above.

Walking a bihash table

A fairly common scenario to build “show” commands involves walking a bihash table. It's simple enough:

```
my_main_t *mm = &my_main;
clib_bihash_8_8_t *h;
void callback_fn (clib_bihash_kv_8_8_t *, void *);

h = &mm->hash_table;

BV(clib_bihash_foreach_key_value_pair) (h, callback_fn, (void *) arg);
```

To nobody's great surprise: clib_bihash_foreach_key_value_pair iterates across the entire table, calling callback_fn with active entries.

Creating a new template instance

Creating a new template is easy. Use one of the existing templates as a model, and make the obvious changes. The hash and key_compare methods are performance-critical in multiple senses.

If the key compare method is slow, every lookup will be slow. If the hash function is slow, same story. If the hash function has poor statistical properties, space efficiency will suffer. In the limit, a bad enough hash function will cause large portions of the table to revert to linear search.

Use of the best available vector unit is well worth the trouble in the hash and key_compare functions.

2.3 Writing VPP Documentation

2.3.1 Building VPP Documents

Overview

These instructions show how the VPP documentation sources are built.

FD.io VPP Documentation uses [reStructuredText](#) (rst) files, which are used by [Sphinx](#). We will also cover how to view your build on Read the Docs in [Using Read the Docs](#).

To build your files, you can either [Create a Virtual Environment using virtualenv](#), which installs all the required applications for you, or you can [Install Sphinx manually](#).

Create a Virtual Environment using virtualenv

For more information on how to use the Python virtual environment check out [Installing packages using pip and virtualenv](#).

Get the Documents

For example start with a clone of the vpp-docs.

```
$ git clone https://gerrit.fd.io/r/vpp
$ cd vpp
```

Install the virtual environment

In your vpp-docs directory, run:

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

Which installs all the required applications into it's own, isolated, virtual environment, so as to not interfere with other builds that may use different versions of software.

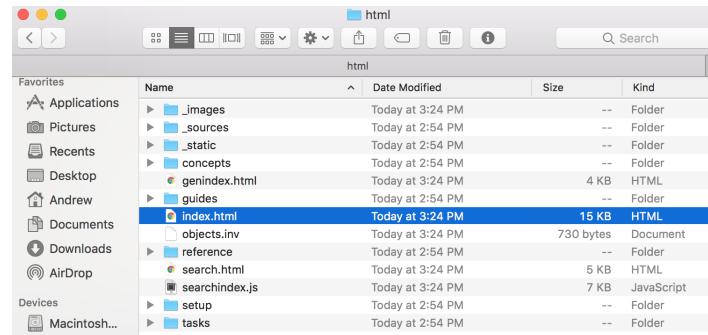
Build the html files

Be sure you are in your vpp-docs/docs directory, since that is where Sphinx will look for your **conf.py** file, and build the **.rst** files into an **index.html** file:

```
$ make html
```

View the results

If there are no errors during the build process, you should now have an **index.html** file in your **vpp/docs/_build/html** directory, which you can then view in your browser.



Whenever you make changes to your **.rst** files that you want to see, repeat this build process.

Note: To exit from the virtual environment execute:

```
$ deactivate
```

Install Sphinx manually

Skip this step if you created a *virtualenv* in the previous step. If you don't want to create a *virtualenv*, you should install Sphinx [here](#), and follow their [getting started guide](#).

Building these files will generate an **index.html** file, which you can then view in your browser to verify and see your file changes.

To *build* your files, make sure you're in your **vpp-docs/docs** directory, where your **conf.py** file is located, and run:

```
$ make html
```

If there are no errors during the build process, you should now have an **index.html** file in your **vpp-docs/docs/_build/html** directory, which you can then view in your browser.

Whenever you make changes to your **.rst** files that you want to see, repeat this build process.

Using Read the Docs

[Read the Docs](#) is a website that “simplifies software documentation by automating building, versioning, and hosting of your docs for you”. Essentially, it accesses your Github repo to generate the **index.html** file, and then displays it on its own *Read the Docs* webpage so others can view your documentation.

Create an account on *Read the Docs* if you haven't already.

Go to your [dashboard](#), and click on “Import a Project”.

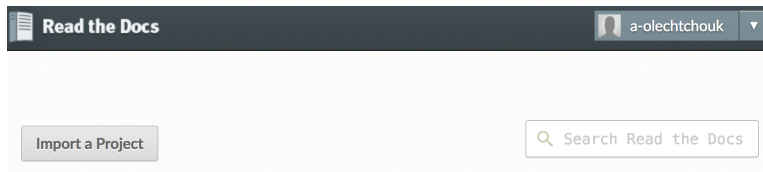
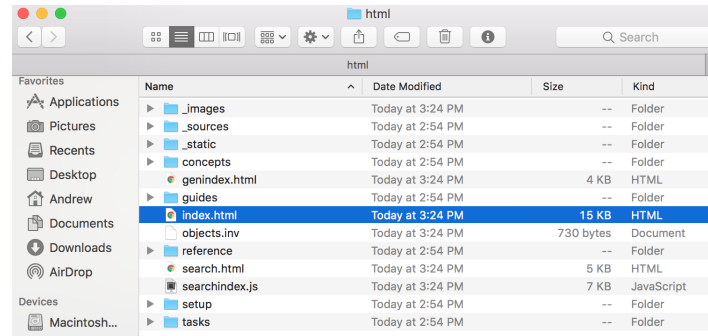
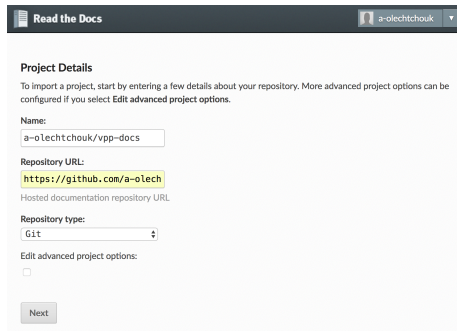


Fig. 1: This will bring you to a page where you can choose to import a repo from your Github account (only if you’ve linked your Github account to your Read the Docs account), or to import a repo manually. In this example, we’ll do it manually. Click “Import Manually”.

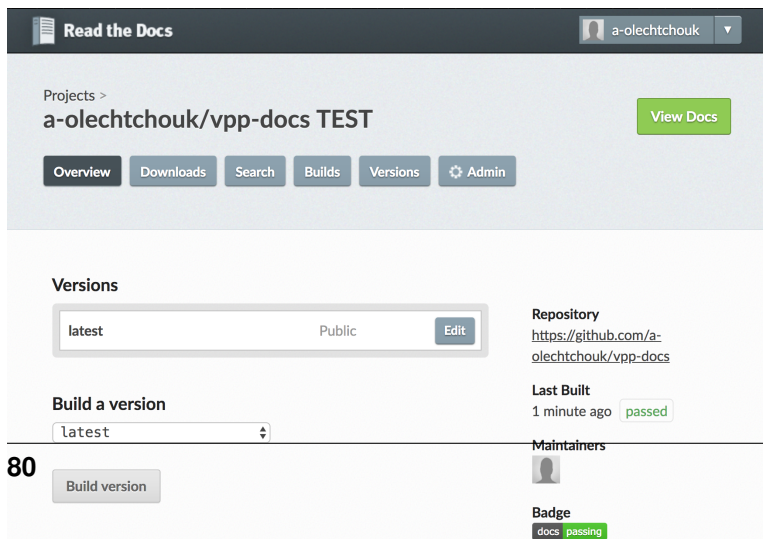
This will bring you to a page that asks for your repo details. Set “Name” to your forked repo name, or whatever you want. Set “Repository URL” to the URL of your forked repo (<https://github.com/YOURUSERNAME/vpp-docs>). “Repository type” should already be selected to “Git”. Then click “Next”.



The screenshot shows the 'Project Details' form on the Read the Docs website. The form is titled 'Project Details' and includes instructions: 'To import a project, start by entering a few details about your repository. More advanced project options can be configured if you select [Edit advanced project options](#).' The form fields are: 'Name' (a-olechtchouk/vpp-docs), 'Repository URL' (https://github.com/a-olech), 'Repository type' (Git), and 'Edit advanced project options' (checkbox). A 'Next' button is at the bottom.

This will bring you to a project page of your repo on Read the Docs. You can confirm it’s the correct repo by checking on the right side of the page the Repository URL.

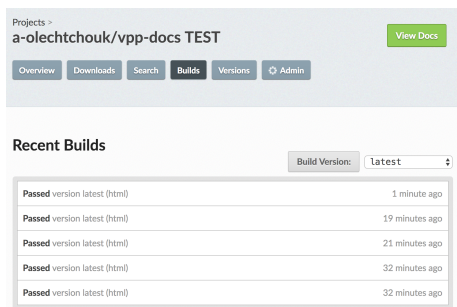
Then click on “Build Version”.



The screenshot shows the project page for 'a-olechtchouk/vpp-docs' on Read the Docs. The page has a header with the Read the Docs logo and the user 'a-olechtchouk'. The main content area shows the project name 'a-olechtchouk/vpp-docs TEST' and a 'View Docs' button. Below this are tabs for 'Overview', 'Downloads', 'Search', 'Builds', 'Versions', and 'Admin'. The 'Versions' tab is selected, showing a list of versions with 'latest' selected. Below the versions list is a 'Build a version' section with a 'Build version' button. On the right side, there is a 'Repository' section with the URL 'https://github.com/a-olechtchouk/vpp-docs', a 'Last Built' section showing '1 minute ago' and 'passed', and a 'Badge' section showing 'docs passing'.

Which takes you to another page showing your recent builds.

Then click on “Build Version:”. This should “Trigger” a build. After about a minute or so you can refresh the page and see that your build “Passed”.



Now on your builds page from the previous image, you can click “View Docs” at the top-right, which will take you a *readthedocs.io* page of your generated build!

2.3.2 Merging FD.io VPP documents

This section describes how to get FD.io VPP documents reviewed and merged.

Git Review

The VPP documents use the gerrit server and git review.

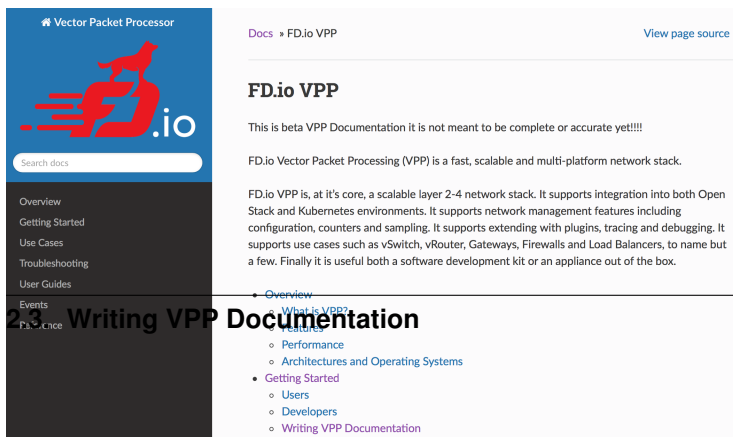
Clone with ssh

To get FD.io VPP documents reviewed the VPP repository should be cloned with ssh.

Use the following to setup you ssh key

```
$ ssh-keygen -t rsa
$ keychain
$ cat ~/.ssh/id_rsa.pub
```

Copy that key to the gerrit server. Then clone the repo with:



New patch

To get a new patch reviewed use the following:

If the patch is a draft use the following:

Note: `git review -D`

To get back to the master:

Existing patch

To modify an existing patch:

2.3.3 Using Read the Docs

[Read the Docs](#) is a website that “simplifies software documentation by automating building, versioning, and hosting of your docs for you”. Essentially, it accesses your Github repo to generate the **index.html** file, and then displays it on its own *Read the Docs* webpage so others can view your documentation.

Create an account on *Read the Docs* if you haven’t already.

Go to your [dashboard](#) , and click on “Import a Project”.

This will bring you to a page that asks for your repo details. Set “Name” to

your forked repo name, or whatever you want. Set “Repository URL” to the URL of your forked repo (<https://github.com/YOURUSERNAME/vpp-docs>).

“Repository type” should already be selected to “Git”. Then click “Next”.

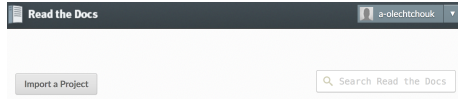
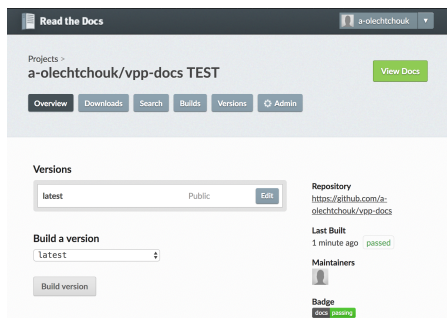


Fig. 2: This will bring you to a page where you can choose to import a repo from your Github account (only if you’ve linked your Github account to your Read the Docs account), or to import a repo manually. In this example, we’ll do it manually. Click “Import Manually”.

This will bring you to a project page of your repo on Read the Docs. You can confirm it’s the correct repo by checking on the right side of the page the Repository URL.

Then click on “Build Version”.



Which takes you to another page showing your recent builds.

Then click on “Build Version:”. This should “Trigger” a build. After about a minute or so you can refresh the page and see that your build “Passed”.

Projects >
a-olechtchouk/vpp-docs TEST [View Docs](#)

[Overview](#) [Downloads](#) [Search](#) [Builds](#) [Versions](#) [Admin](#)

Recent Builds

Build Version:

Passed version latest (html)	1 minute ago
Passed version latest (html)	19 minutes ago
Passed version latest (html)	21 minutes ago
Passed version latest (html)	32 minutes ago
Passed version latest (html)	32 minutes ago

Now on your builds page from the previous image, you can click “View Docs” at the top-right, which will take you a *readthedocs.io* page of your generated build!

2.3.4 reStructured Text Style Guide

Most of these documents are written in reStructured Text (rst). This chapter describes some of the Sphinx Markup Constructs used in these documents. The Sphinx style guide can be found at: [Sphinx Style Guide](#). For a more detailed list of Sphinx Markup Constructs please refer to: [Sphinx Markup Constructs](#).

This document is also an example of a directory structure for a document that spans multiple pages. Notice we have the file **index.rst** and the then documents that are

referenced in **index.rst**. The referenced documents are shown at the bottom of this page.

A label is shown at the top of this page. Then the first construct describes the document title **FD.io Style Guide**. Text usually follows under each title or heading.

A **Table of Contents** structure is shown below. Using **toctree** in this way will show the headings in a nicely in the generated documents.

Heading 1

This is the top heading level. More levels are shown below.

Heading 2

Heading 3

Heading 4

Heading 5

Bullets, Bold and Italics

Bold text can be show with **Bold Text**, Italics with *Italic text*. Bullets like so:

- Bullet 1
 - Bullet 2
1. Numbered Bullet 1
 2. Numbered Bullet 2

Notes

A note can be used to describe something not in the normal flow of the paragraph. This is an example of a note.

Note: Using **git commit** after adding your files saves a “Snapshot” of them, so it’s very hard to lose your work if you *commit often*.

Code Blocks

This paragraph describes how to do **Console Commands**. When showing VPP commands it is reccomended that the command be executed from the linux console as shown. The Highlighting in the final documents shows up nicely this way.

```
$ sudo bash
# vppctl show interface
      Name                               Idx      State      Counter      Count
TenGigabitEthernet86/0/0                1        up        rx packets    6569213
                                           rx bytes     9928352943
                                           tx packets    50384
                                           tx bytes     3329279
TenGigabitEthernet86/0/1                2        down
VirtualEthernet0/0/0                    3        up        rx packets    50384
                                           rx bytes     3329279
                                           tx packets    6569213
                                           tx bytes     9928352943
                                           drops        1498
local0                                  0        down
#
```

The **code-block** construct is also used for code samples. The following shows how to include a block of “C” code.

```
#include <vlib/unix/unix.h>
abf_policy_t *
abf_policy_get (u32 index)
{
    return (pool_elt_at_index (abf_policy_pool, index));
}
```

Diffs are generated in the final docs nicely with “:” at the end of the description like so:

```
diff --git a/src/vpp/vnet/main.c b/src/vpp/vnet/main.c
index 6e136e19..69189c93 100644
--- a/src/vpp/vnet/main.c
+++ b/src/vpp/vnet/main.c
@@ -18,6 +18,8 @@
    #include <vlib/unix/unix.h>
    #include <vnet/plugin/plugin.h>
    #include <vnet/ethernet/ethernet.h>
+   #include <vnet/ip/ip4_packet.h>
+   #include <vnet/ip/format.h>
    #include <vpp/app/version.h>
    #include <vpp/api/vpe_msg_enum.h>
    #include <limits.h>
@@ -400,6 +402,63 @@ VLIB_CLI_COMMAND (test_crash_command, static) = {

    #endif
```

Tables

There are two types of tables with different syntax, [Grid Tables](#), and [Simple Tables](#).

Grid Tables

[Grid Tables](#) are described with a visual grid made up of the characters “-“, “=“, “|”, and “+”. The hyphen (“-“) is used for horizontal lines (row separators). The equals sign (“=“) may be used to separate optional header rows from the table body. The vertical bar (“|”) is used for vertical lines (column separators). The plus sign (“+“) is used for intersections of horizontal and vertical lines.

Here is example code for a grid table in a *.rst* file:

```
+-----+-----+-----+-----+
| Header row, column 1 | Header 2 | Header 3 | Header 4 |
| (header rows optional) | | | |
+=====+=====+=====+=====+
| body row 1, column 1 | column 2 | column 3 | column 4 |
+-----+-----+-----+-----+
| body row 2 | Cells may span columns. | | |
+-----+-----+-----+-----+
| body row 3 | Cells may | - Table cells | |
+-----+-----+-----+-----+
| body row 4 | span rows. | - contain | |
| | | - body elements. | |
+-----+-----+-----+-----+
```

This example code generates a grid table that looks like this:

Header row, column 1 (header rows optional)	Header 2	Header 3	Header 4
body row 1, column 1	column 2	column 3	column 4
body row 2	Cells may span columns.		
body row 3	Cells may span rows.	<ul style="list-style-type: none">• Table cells• contain• body elements.	
body row 4			

Simple Tables

Simple tables are described with horizontal borders made up of “=” and “-” characters. The equals sign (“=”) is used for top and bottom table borders, and to separate optional header rows from the table body. The hyphen (“-”) is used to indicate column spans in a single row by underlining the joined columns, and may optionally be used to explicitly and/or visually separate rows.

Simple tables are “simpler” to create than grid tables, but are more limited.

Here is example code for a simple table in a *.rst* file.

```
=====
  A      B      A and B
=====
False  False  False
True   False  False
False  True   False
True   True   True
=====
```

This example code generates a simple table that looks like this:

A	B	A and B
False	False	False
True	False	False
False	True	False
True	True	True

Labels, References

A link or reference to other paragraphs within these documents can be done with following construct.

In this example the reference points the label **showintcommand**. The label **styleguide03** is shown at the top of this page. A label used in this way must be above a heading or title.

Show Interface command.

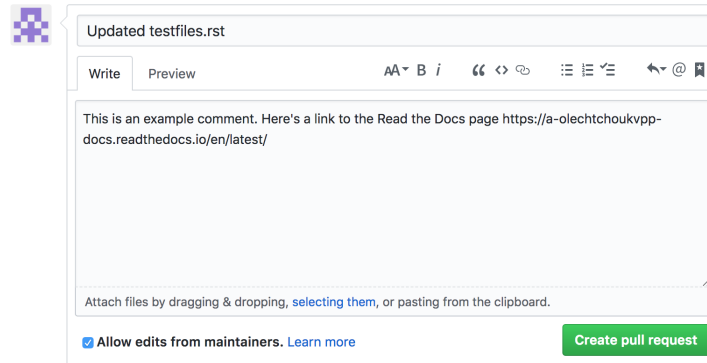
External Links

An external link is done with the following construct:

[Sphinx Markup Constructs](#)

Images

Images should be placed in the directory docs/_images. They can then be referenced with following construct. This is the image created to show a pull request.



Including a file

A complete file should be included with the following construct. It is recommended it be included with it's own .rst file describing the file included. This is an example of an xml file is included.

An XML File

An example of an XML file.

```
<domain type='kvm' id='54'>
  <name>iperf-server</name>
  <memory unit='KiB'>1048576</memory>
  <currentMemory unit='KiB'>1048576</currentMemory>
  <memoryBacking>
    <hugepages>
      <page size='2048' unit='KiB' />
    </hugepages>
  </memoryBacking>
  <vcpu placement='static'>1</vcpu>
  <resource>
    <partition>/machine</partition>
  </resource>
  <os>
    <type arch='x86_64' machine='pc-i440fx-xenial'>hvm</type>
    <boot dev='hd' />
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <cpu mode='host-model'>
    <model fallback='allow'></model>
    <numa>
      <cell id='0' cpus='0' memory='262144' unit='KiB' memAccess='shared' />
    </numa>
  </cpu>
</domain>
```

(continues on next page)

(continued from previous page)

```

<clock offset='utc'>
  <timer name='rtc' tickpolicy='catchup' />
  <timer name='pit' tickpolicy='delay' />
  <timer name='hpet' present='no' />
</clock>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<pm>
  <suspend-to-mem enabled='no' />
  <suspend-to-disk enabled='no' />
</pm>
<devices>
  <emulator>/usr/bin/kvm</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2' />
    <source file='/tmp/xenial-mod.img' />
    <backingStore />
    <target dev='vda' bus='virtio' />
    <alias name='virtio-disk0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0' />
  </disk>
  <disk type='file' device='cdrom'>
    <driver name='qemu' type='raw' />
    <source file='/scratch/jdenisco/sae/configs/cloud-config.iso' />
    <backingStore />
    <target dev='hda' bus='ide' />
    <readonly />
    <alias name='ide0-0-0' />
    <address type='drive' controller='0' bus='0' target='0' unit='0' />
  </disk>
  <controller type='usb' index='0' model='ich9-ehci1'>
    <alias name='usb' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x7' />
  </controller>
  <controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0' />
  </controller>
  <controller type='ide' index='0'>
    <alias name='ide' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1' />
  </controller>
  <controller type='virtio-serial' index='0'>
    <alias name='virtio-serial0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0' />
  </controller>
  <interface type='vhostuser'>
    <mac address='52:54:00:4c:47:f2' />
    <source type='unix' path='/tmp/vm00.sock' mode='server' />
    <model type='virtio' />
    <alias name='net1' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0' />
  </interface>
  <serial type='pty'>
    <source path='/dev/pts/2' />
    <target port='0' />
    <alias name='serial0' />

```

(continues on next page)

(continued from previous page)

```

</serial>
<console type='pty' tty='/dev/pts/2'>
  <source path='/dev/pts/2' />
  <target type='serial' port='0' />
  <alias name='serial0' />
</console>
<input type='mouse' bus='ps2' />
<input type='keyboard' bus='ps2' />
<graphics type='vnc' port='5900' autoport='yes' listen='127.0.0.1'>
  <listen type='address' address='127.0.0.1' />
</graphics>
<memballoon model='virtio'>
  <alias name='balloon0' />
  <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0' />
</memballoon>
</devices>
<seclabel type='dynamic' model='apparmor' relabel='yes'>
  <label>libvirt-2c4c9317-c7a5-4b37-b789-386ccda7348a</label>
  <imagelabel>libvirt-2c4c9317-c7a5-4b37-b789-386ccda7348a</imagelabel>
</seclabel>
</domain>

```

Raw HTML

An html frame can be included with the following construct. It is recommended that references to raw html be included with it's own .rst file.

Raw HTML Example

This example shows how to include include a CSIT performance graph.

2.3.5 Markdown Style Guide

Most of these documents are written using *reStructured Text Style Guide* (rst), but pages can also be written in Markdown. This chapter describes some constructs used to write these documents. For a more detailed description of Markdown refer to [Markdown Wikipedia](#)

Heading 1

This is the top heading level. More levels are shown below.

Heading 2

Heading 3

Heading 4

Heading 5

Bullets, Bold and Italics

Bold text can be show with **Bold Text**, Italics with *Italic text*. Bullets like so:

- Bullet 1
- Bullet 2

Code Blocks

This paragraph describes how to do **Console Commands**. When showing VPP commands it is reccomended that the command be executed from the linux console as shown. The Highlighting in the final documents shows up nicely this way.

```
$ sudo bash
# vppctl show interface
```

Name	Idx	State	Counter	Count
TenGigabitEthernet86/0/0	1	up	rx packets	6569213
			rx bytes	9928352943
			tx packets	50384
			tx bytes	3329279
TenGigabitEthernet86/0/1	2	down		
VirtualEthernet0/0/0	3	up	rx packets	50384
			rx bytes	3329279
			tx packets	6569213
			tx bytes	9928352943
			drops	1498
local0	0	down		

```
#
```

The **code-block** construct is also used for code samples. The following shows how to include a block of “C” code.

```
#include <vlib/unix/unix.h>
abf_policy_t *
abf_policy_get (u32 index)
{
    return (pool_elt_at_index (abf_policy_pool, index));
}
```

Diffs are generated in the final docs nicely with “:” at the end of the description like so:

```
diff --git a/src/vpp/vnet/main.c b/src/vpp/vnet/main.c
index 6e136e19..69189c93 100644
--- a/src/vpp/vnet/main.c
+++ b/src/vpp/vnet/main.c
@@ -18,6 +18,8 @@
     #include <vlib/unix/unix.h>
```

(continues on next page)

(continued from previous page)

```
#include <vnet/plugin/plugin.h>
#include <vnet/ethernet/ethernet.h>
+#include <vnet/ip/ip4_packet.h>
+#include <vnet/ip/format.h>
#include <vpp/app/version.h>
#include <vpp/api/vpe_msg_enum.h>
#include <limits.h>
@@ -400,6 +402,63 @@ VLIB_CLI_COMMAND (test_crash_command, static) = {

#endif
```

Labels, References

A link or reference to other paragraphs within these documents can be done with following construct.

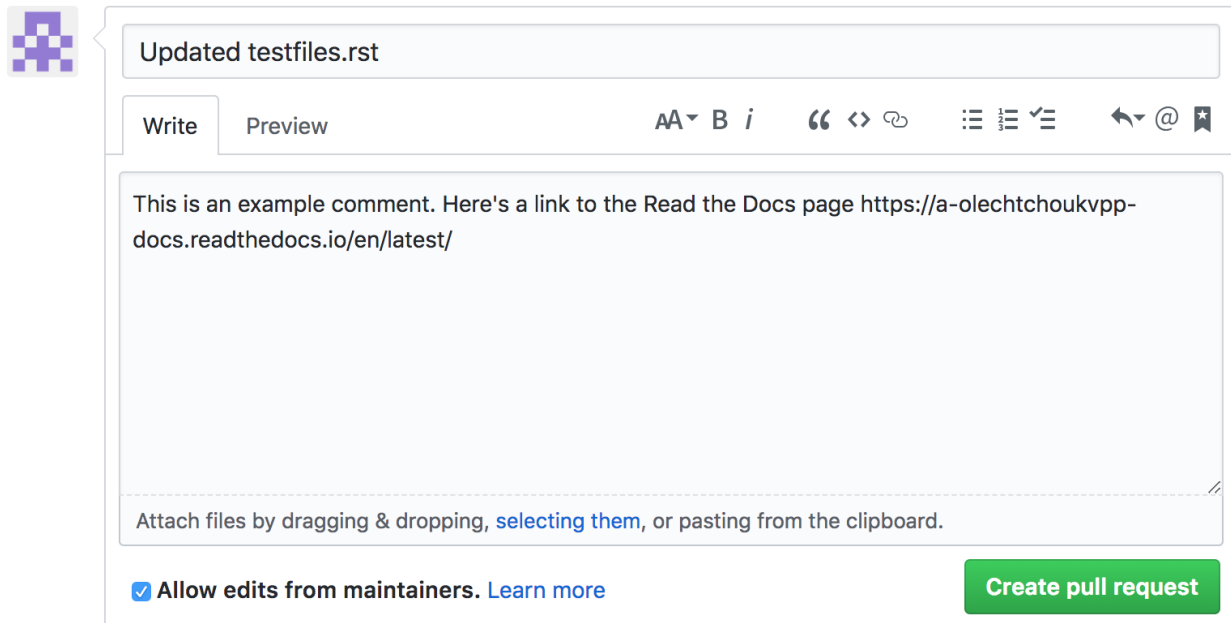
External Links

An external link is done with the following construct:

[Sphinx Markup Constructs](#)

Images

Images should be placed in the directory docs/_images. They can then be referenced with following construct. This is the image created to show a pull request.



2.3.6 To Do

This section describes pieces of these documents that need some work.

All Sections

All the sections need to be spell checked.

Checked for guidelines.

FD.io VPP with Virtual Machines

Topology

Get a better topology picture.

Creating the Virtual Machine

The XML file refers to an iso image, come up with a procedure to build that image. That should work when used with a cloud image. It should also be mentioned where to get a cloud image.

It is mentioned that a queue size of 256 is not large enough. Come up with a procedure to change the queue size.

Users

The getting started users guide needs an overview

Command Line Reference

This section should be references to the doxygen links. The doxygen links will need to be cleaned up. This section could be a quick reference only using commands we have tested.

progressivevpp

This section needs work. It needs to be split up and tested.

vSwitch/vRouter

Needs some instructions or be removed.

Downloading the jvpp jar

Not sure what value this provides.

2.3.7 Github Repository

The github repository is no longer being used. The gerrit server is being used.

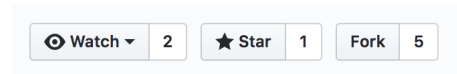
To use the gerrit repository refer to *Merging FD.io VPP documents*.

Overview

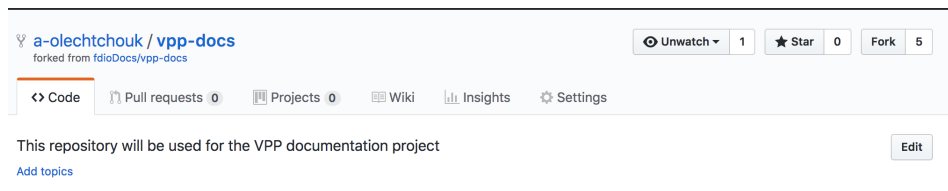
This section will cover how to fork your own branch of the [fdioDocs/vpp-docs](#) repository, clone that repo locally to your computer, make changes to it, and how to issue a pull request when you want your changes to be reflected on the main repo.

Forking your own branch

In your browser, navigate to the repo you want to branch off of. In this case, the [fdioDocs/vpp-docs](#) repo. At the top right of the page you should see this:



Click on “Fork”, and then a pop-up should appear where you should then click your Github username. Once this is done, it should automatically take you to the Github page where your new branch is located, just like in the image below.



Now your **own branch** can be **cloned** to your computer using the URL (<https://github.com/YOURUSERNAME/vpp-docs>) of the Github page where your branch is located.

Creating a local repository

Now that you have your own branch of the main repository on Github, you can store it locally on your computer. In your shell, navigate to the directory where you want to store your branch/repo. Then execute:

```
$ git clone https://github.com/YOURUSERNAME/vpp-docs
```

This will create a directory on your computer named **vpp-docs**, the name of the repo.

Now that your branch is on your computer, you can modify and build files however you wish.

If you are not on the master branch, move to it.

```
$ git checkout master
```

Keeping your files in sync with the main repo

The following talks about remote branches, but keep in mind that there are currently *two* branches, your local “master” branch (on your computer), and your remote “origin or origin/master” branch (the one you created using “Fork” on the Github website).

You can view your *remote* repositories with:

```
$ git remote -v
```

At this point, you may only see the remote branch that you cloned from.

```
Macintosh:docs Andrew$ git remote -v
origin  https://github.com/a-olechtchouk/vpp-docs (fetch)
origin  https://github.com/a-olechtchouk/vpp-docs (push)
```

Now you want to create a new remote repository of the main vpp-docs repo (naming it upstream).

```
$ git remote add upstream https://github.com/fdioDocs/vpp-docs
```

You can verify that you have added a remote repo using the previous **git remote -v** command.

```
$ git remote -v
origin  https://github.com/a-olechtchouk/vpp-docs (fetch)
origin  https://github.com/a-olechtchouk/vpp-docs (push)
upstream https://github.com/fdioDocs/vpp-docs (fetch)
upstream https://github.com/fdioDocs/vpp-docs (push)
```

If there have been any changes to files in the main repo (hopefully not the same files you were working on!), you want to make sure your local branch is in sync with them.

To do so, fetch any changes that the main repo has made, and then merge them into your local master branch using:

```
$ git fetch upstream
$ git merge upstream/master
```

Note: This is optional, so don't do these commands if you just want one local branch!!!

You may want to have multiple branches, where each branch has its own different features, allowing you to have multiple pull requests out at a time. To create a new local branch:

```
$ git checkout -b cleanup-01
$ git branch
* cleanup-01
  master
  overview
```

Now you can redo the previous steps **for** "Keeping your files in sync with the main repo" **for** your newly created local branch, and **then** depending on which branch you want, **to** send out a pull request **for**, proceed below.

Pushing to your branch

Now that your files are in sync, you want to add modified files, commit, and push them from *your local branch* to your *personal remote branch* (not the main fdioDocs repo).

To check the status of your files, run:

```
$ git status
```

In the output example below, I deleted gettingsources.rst, made changes to index.rst and pushingapatch.rst, and have created a new file called buildingrst.rst.

```

Macintosh:docs Andrew$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    tasks/writingdocs/gettingsources.rst

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   tasks/writingdocs/index.rst
    modified:   tasks/writingdocs/pushingapatch.rst

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    tasks/writingdocs/buildingrst.rst

```

To add files (use **git add -A** to add all modified files):

```
$ git add FILENAME1 FILENAME2
```

Commit and push using:

```
$ git commit -m 'A descriptive commit message for two files.'
```

Push your changes for the branch where your changes were made

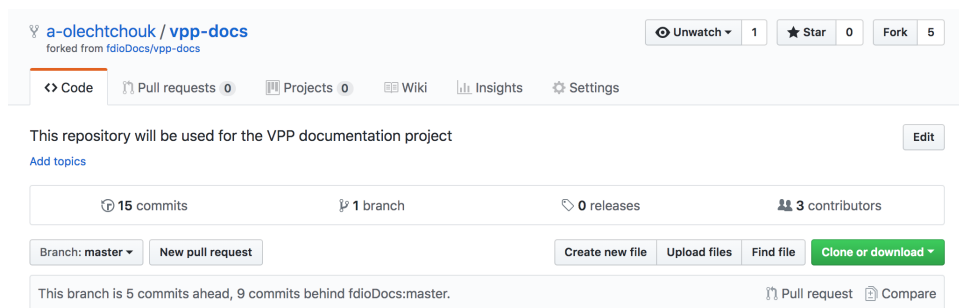
```
$ git push origin <branch name>
```

Here, your personal remote branch is “origin” and your local branch is “master”.

Note: Using **git commit** after adding your files saves a “Snapshot” of them, so it’s very hard to lose your work if you *commit often*.

Initiating a pull request (Code review)

Once you’ve pushed your changes to your remote branch, go to your remote branch on Github (<https://github.com/YOURUSERNAME/vpp-docs>), and click on “New pull request”.



This will bring you to a “Comparing changes” page. Click “Create new pull request”.

Create new pull request

Which will open up text fields to add information to your pull request.

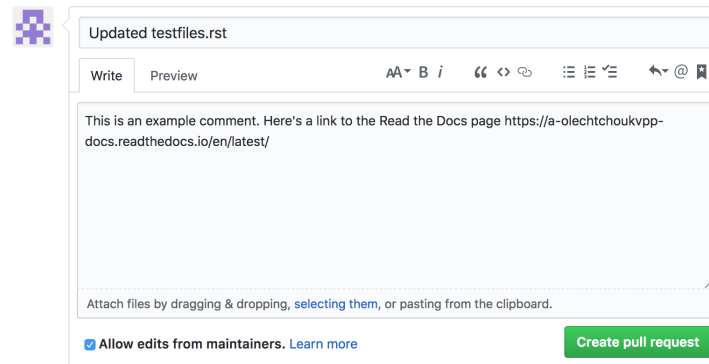


Fig. 3: Then finally click “Create pull request” to complete the pull request.

Your documents will be reviewed. To this same branch make the changes requested from the review and then push your new changes. There is no need to create another pull request.

```
$ git commit -m 'A descriptive commit message for the new changes'
$ git push origin <branch name>
```

Additional Git commands

You may find some of these Git commands useful:

Use **git diff** to quickly show the file changes and repo differences of your commits.

Use **git rm FILENAME** to stop tracking a file and to remove it from your remote branch and local directory. Use flag **-r** to remove folders/directories. E.g (**git rm -r oldfolder**)

This chapter contains a sample of the many ways FD.io VPP can be used. It is by no means an extensive list, but should give a sampling of the many features contained in FD.io VPP.

3.1 FD.io VPP with Containers

This section will cover connecting two Linux containers with VPP. A container is essentially a more efficient and faster VM, due to the fact that a container does not simulate a separate kernel and hardware. You can read more about [Linux containers here](#).

3.1.1 Creating Containers

First you should have root privileges:

```
$ sudo bash
```

Then install packages for containers such as lxc:

```
# apt-get install bridge-utils lxc
```

As quoted from the [lxc.conf manpage](#), “container configuration is held in the config stored in the container’s directory. A basic configuration is generated at container creation time with the default’s recommended for the chosen template as well as extra default keys coming from the default.conf file.”

“That *default.conf* file is either located at `/etc/lxc/default.conf` or for unprivileged containers at `~/.config/lxc/default.conf`.”

Since we want to ping between two containers, we’ll need to **add to this file**.

Look at the contents of *default.conf*, which should initially look like this:

```
# cat /etc/lxc/default.conf
lxc.network.type = veth
lxc.network.link = lxcbr0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:xx:xx:xx
```

As you can see, by default there is one veth interface.

Now you will *append to this file* so that each container you create will have an interface for a Linux bridge and an unconsumed second interface.

You can do this by piping *echo* output into *tee*, where each line is separated with a newline character `\n` as shown below. Alternatively, you can manually add to this file with a text editor such as *vi*, but make sure you have root privileges.

```
# echo -e "lxc.network.name = veth0\nlxc.network.type = veth\nlxc.network.name = veth_\n↪link1" | sudo tee -a /etc/lxc/default.conf
```

Inspect the contents again to verify the file was indeed modified:

```
# cat /etc/lxc/default.conf
lxc.network.type = veth
lxc.network.link = lxcbr0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:xx:xx:xx
lxc.network.name = veth0
lxc.network.type = veth
lxc.network.name = veth_link1
```

After this, we're ready to create the containers.

Creates an Ubuntu Xenial container named “cone”.

```
# lxc-create -t download -n cone -- --dist ubuntu --release xenial --arch amd64 --
↪keyserver hkp://p80.pool.sks-keyservers.net:80
```

If successful, you'll get an output similar to this:

```
You just created an Ubuntu xenial amd64 (20180625_07:42) container.

To enable SSH, run: apt install openssh-server
No default root or user password are set by LXC.
```

Make another container “ctwo”.

```
# lxc-create -t download -n ctwo -- --dist ubuntu --release xenial --arch amd64 --
↪keyserver hkp://p80.pool.sks-keyservers.net:80
```

List your containers to verify they exist:

```
# lxc-ls
cone ctwo
```

Start the first container:

```
# lxc-start --name cone
```

And verify its running:


```
# lxc-ls --fancy
NAME STATE   AUTOSTART GROUPS IPV4 IPV6
cone RUNNING 0         -    -    -
ctwo STOPPED 0         -    -    -
```

Note: Here are some `lxc` container commands you may find useful:

```
sudo lxc-ls --fancy
sudo lxc-start --name u1 --daemon
sudo lxc-info --name u1
sudo lxc-stop --name u1
sudo lxc-destroy --name u1
```

3.1.2 Container packages

Now we can go into container *cone* and install prerequisites such as VPP, and perform some additional commands:

To enter our container via the shell, type:

```
# lxc-attach -n cone
root@cone:/#
```

Run the linux DHCP setup and install VPP:

```
root@cone:/# resolvconf -d eth0
root@cone:/# dhclient
root@cone:/# apt-get install -y wget
root@cone:/# echo "deb [trusted=yes] https://nexus.fd.io/content/repositories/fd.io.
↳ubuntu.xenial.main/ ./" | sudo tee -a /etc/apt/sources.list.d/99fd.io.list
root@cone:/# apt-get update
root@cone:/# apt-get install -y --force-yes vpp
root@cone:/# sh -c 'echo "\\ndpdk {\\n    no-pci\\n}\\n" >> /etc/vpp/startup.conf'
```

After this is done, start VPP in this container:

```
root@cone:/# service vpp start
```

Exit this container with the **exit** command (you *may* need to run **exit** twice):

```
root@cone:/# exit
exit
root@cone:/# exit
exit
root@localhost:~#
```

Repeat the container setup on this page for the second container **ctwo**. Go to the end of the previous page if you forgot how to start a container.

3.1.3 Connecting the two Containers

Now for connecting these two linux containers to VPP and pinging between them.

Enter container *cone*, and check the current network configuration:

```

root@ccone:/# ip -o a
1: lo      inet 127.0.0.1/8 scope host lo\          valid_lft forever preferred_lft forever
1: lo      inet6 ::1/128 scope host \          valid_lft forever preferred_lft forever
30: veth0   inet 10.0.3.157/24 brd 10.0.3.255 scope global veth0\          valid_lft_
↪forever preferred_lft forever
30: veth0   inet6 fe80::216:3eff:fee2:d0ba/64 scope link \          valid_lft forever_
↪preferred_lft forever
32: veth_link1 inet6 fe80::2c9d:83ff:fe33:37e/64 scope link \          valid_lft_
↪forever preferred_lft forever

```

You can see that there are three network interfaces, *lo*, *veth0*, and *veth_link1*.

Notice that *veth_link1* has no assigned IP.

Check if the interfaces are down or up:

```

root@ccone:/# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT_
↪group default qlen 1
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
30: veth0@if31: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP_
↪mode DEFAULT group default qlen 1000
   link/ether 00:16:3e:e2:d0:ba brd ff:ff:ff:ff:ff:ff link-netnsid 0
32: veth_link1@if33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state_
↪UP mode DEFAULT group default qlen 1000
   link/ether 2e:9d:83:33:03:7e brd ff:ff:ff:ff:ff:ff link-netnsid 0

```

Note: Take note of the network index for **veth_link1**. In our case, it 32, and its parent index (the host machine, not the containers) is 33, shown by **veth_link1@if33**. Yours will most likely be different, but **please take note of these index's**.

Make sure your loopback interface is up, and assign an IP and gateway to *veth_link1*.

```

root@ccone:/# ip link set dev lo up
root@ccone:/# ip addr add 172.16.1.2/24 dev veth_link1
root@ccone:/# ip link set dev veth_link1 up
root@ccone:/# dhclient -r
root@ccone:/# ip route add default via 172.16.1.1 dev veth_link1

```

Here, the IP is 172.16.1.2/24 and the gateway is 172.16.1.1.

Run some commands to verify the changes:

```

root@ccone:/# ip -o a
1: lo      inet 127.0.0.1/8 scope host lo\          valid_lft forever preferred_lft forever
1: lo      inet6 ::1/128 scope host \          valid_lft forever preferred_lft forever
30: veth0   inet6 fe80::216:3eff:fee2:d0ba/64 scope link \          valid_lft forever_
↪preferred_lft forever
32: veth_link1 inet 172.16.1.2/24 scope global veth_link1\          valid_lft forever_
↪preferred_lft forever
32: veth_link1 inet6 fe80::2c9d:83ff:fe33:37e/64 scope link \          valid_lft_
↪forever preferred_lft forever

root@ccone:/# route
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface

```

(continues on next page)

(continued from previous page)

default	172.16.1.1	0.0.0.0	UG	0	0	0 veth_link1
172.16.1.0	*	255.255.255.0	U	0	0	0 veth_link1

We see that the IP has been assigned, as well as our default gateway.

Now exit this container and repeat this process with container *ctwo*, except with IP 172.16.2.2/24 and gateway 172.16.2.1.

After that's done for *both* containers, exit from the container if you're in one:

```
root@ctwo:/# exit
exit
root@localhost:~#
```

In the machine running the containers, run **ip link** to see the host *veth* network interfaces, and their link with their respective *container veth's*.

```
root@localhost:~# ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode_
   ↪DEFAULT group default qlen 1000
   link/ether 08:00:27:33:82:8a brd ff:ff:ff:ff:ff:ff
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode_
   ↪DEFAULT group default qlen 1000
   link/ether 08:00:27:d9:9f:ac brd ff:ff:ff:ff:ff:ff
4: enp0s9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode_
   ↪DEFAULT group default qlen 1000
   link/ether 08:00:27:78:84:9d brd ff:ff:ff:ff:ff:ff
5: lxcbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state UP mode_
   ↪DEFAULT group default qlen 1000
   link/ether 00:16:3e:00:00:00 brd ff:ff:ff:ff:ff:ff
19: veth0C2FL7@if18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop master_
   ↪lxcbr0 state UP mode DEFAULT group default qlen 1000
   link/ether fe:0d:da:90:c1:65 brd ff:ff:ff:ff:ff:ff link-netnsid 1
21: veth8NA72P@if20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state_
   ↪UP mode DEFAULT group default qlen 1000
   link/ether fe:1c:9e:01:9f:82 brd ff:ff:ff:ff:ff:ff link-netnsid 1
31: vethXQMY4C@if30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop master_
   ↪lxcbr0 state UP mode DEFAULT group default qlen 1000
   link/ether fe:9a:d9:29:40:bb brd ff:ff:ff:ff:ff:ff link-netnsid 0
33: vethQL7KOC@if32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state_
   ↪UP mode DEFAULT group default qlen 1000
   link/ether fe:ed:89:54:47:a2 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Remember our network interface index 32 in *cone* from this [note](#)? We can see at the bottom the name of the 33rd index **vethQL7KOC@if32**. Keep note of this network interface name for the veth connected to *cone* (ex. vethQL7KOC), and the other network interface name for *ctwo*.

With VPP in the host machine, show current VPP interfaces:

```
root@localhost:~# vppctl show inter
      Name      Idx  State  MTU (L3/IP4/IP6/MPLS)  Counter
↪Count
local0          0   down      0/0/0/0
```

Which should only output local0.

Based on the names of the network interfaces discussed previously, which are specific to my systems, we can create VPP host-interfaces:

```
root@localhost:~# vppctl create host-interface name vethQL7K0C
root@localhost:~# vppctl create host-interface name veth8NA72P
```

Verify they have been set up properly:

```
root@localhost:~# vppctl show inter
```

Name	Idx	State	MTU (L3/IP4/IP6/MPLS)	Counter
↔Count				
host-vethQL7K0C	1	down	9000/0/0/0	
host-veth8NA72P	2	down	9000/0/0/0	
local0	0	down	0/0/0/0	

Which should output *three network interfaces*, local0, and the other two host network interfaces linked to the container veth's.

Set their state to up:

```
root@localhost:~# vppctl set interface state host-vethQL7K0C up
root@localhost:~# vppctl set interface state host-veth8NA72P up
```

Verify they are now up:

```
root@localhost:~# vppctl show inter
```

Name	Idx	State	MTU (L3/IP4/IP6/MPLS)	Counter
↔Count				
host-vethQL7K0C	1	up	9000/0/0/0	
host-veth8NA72P	2	up	9000/0/0/0	
local0	0	down	0/0/0/0	

Add IP addresses for the other end of each veth link:

```
root@localhost:~# vppctl set interface ip address host-vethQL7K0C 172.16.1.1/24
root@localhost:~# vppctl set interface ip address host-veth8NA72P 172.16.2.1/24
```

Verify the addresses are set properly by looking at the L3 table:

```
root@localhost:~# vppctl show inter addr
host-vethQL7K0C (up):
  L3 172.16.1.1/24
host-veth8NA72P (up):
  L3 172.16.2.1/24
local0 (dn):
```

Or looking at the FIB by doing:

```
root@localhost:~# vppctl show ip fib
ipv4-VRF:0, fib_index:0, flow hash:[src dst sport dport proto ] locks:[src:plugin-
↔hi:2, src:default-route:1, ]
0.0.0.0/0
  unicast-ip4-chain
  [@0]: dpo-load-balance: [proto:ip4 index:1 buckets:1 uRPF:0 to:[0:0]]
  [0] [@0]: dpo-drop ip4
0.0.0.0/32
  unicast-ip4-chain
  [@0]: dpo-load-balance: [proto:ip4 index:2 buckets:1 uRPF:1 to:[0:0]]
```

(continues on next page)

(continued from previous page)

```

    [0] [0]: dpo-drop ip4
172.16.1.0/32
    unicast-ip4-chain
    [0]: dpo-load-balance: [proto:ip4 index:10 buckets:1 uRPF:9 to:[0:0]]
    [0] [0]: dpo-drop ip4
172.16.1.0/24
    unicast-ip4-chain
    [0]: dpo-load-balance: [proto:ip4 index:9 buckets:1 uRPF:8 to:[0:0]]
    [0] [0]: ipv4-glean: host-vethQL7K0C: mtu:9000 ffffffff02fec953f98c0806
172.16.1.1/32
    unicast-ip4-chain
    [0]: dpo-load-balance: [proto:ip4 index:12 buckets:1 uRPF:13 to:[0:0]]
    [0] [0]: dpo-receive: 172.16.1.1 on host-vethQL7K0C
172.16.1.255/32
    unicast-ip4-chain
    [0]: dpo-load-balance: [proto:ip4 index:11 buckets:1 uRPF:11 to:[0:0]]
    [0] [0]: dpo-drop ip4
172.16.2.0/32
    unicast-ip4-chain
    [0]: dpo-load-balance: [proto:ip4 index:14 buckets:1 uRPF:15 to:[0:0]]
    [0] [0]: dpo-drop ip4
172.16.2.0/24
    unicast-ip4-chain
    [0]: dpo-load-balance: [proto:ip4 index:13 buckets:1 uRPF:14 to:[0:0]]
    [0] [0]: ipv4-glean: host-veth8NA72P: mtu:9000 ffffffff02fe305400e80806
172.16.2.1/32
    unicast-ip4-chain
    [0]: dpo-load-balance: [proto:ip4 index:16 buckets:1 uRPF:19 to:[0:0]]
    [0] [0]: dpo-receive: 172.16.2.1 on host-veth8NA72P
172.16.2.255/32
    unicast-ip4-chain
    [0]: dpo-load-balance: [proto:ip4 index:15 buckets:1 uRPF:17 to:[0:0]]
    [0] [0]: dpo-drop ip4
224.0.0.0/4
    unicast-ip4-chain
    [0]: dpo-load-balance: [proto:ip4 index:4 buckets:1 uRPF:3 to:[0:0]]
    [0] [0]: dpo-drop ip4
240.0.0.0/4
    unicast-ip4-chain
    [0]: dpo-load-balance: [proto:ip4 index:3 buckets:1 uRPF:2 to:[0:0]]
    [0] [0]: dpo-drop ip4
255.255.255.255/32
    unicast-ip4-chain
    [0]: dpo-load-balance: [proto:ip4 index:5 buckets:1 uRPF:4 to:[0:0]]
    [0] [0]: dpo-drop ip4

```

At long last you probably want to see some pings:

```

root@localhost:~# lxc-attach -n cone -- ping -c3 172.16.2.2
PING 172.16.2.2 (172.16.2.2) 56(84) bytes of data.
64 bytes from 172.16.2.2: icmp_seq=1 ttl=63 time=0.102 ms
64 bytes from 172.16.2.2: icmp_seq=2 ttl=63 time=0.189 ms
64 bytes from 172.16.2.2: icmp_seq=3 ttl=63 time=0.150 ms

--- 172.16.2.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.102/0.147/0.189/0.035 ms

```

(continues on next page)

(continued from previous page)

```

root@localhost:~# lxc-attach -n ctwo -- ping -c3 172.16.1.2
PING 172.16.1.2 (172.16.1.2) 56(84) bytes of data.
64 bytes from 172.16.1.2: icmp_seq=1 ttl=63 time=0.111 ms
64 bytes from 172.16.1.2: icmp_seq=2 ttl=63 time=0.089 ms
64 bytes from 172.16.1.2: icmp_seq=3 ttl=63 time=0.096 ms

--- 172.16.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.089/0.098/0.111/0.014 ms

```

Which should send/recieve three packets for each command.

This is the end of this guide. Great work!

3.2 FD.io VPP with Virtual Machines

This chapter will describe how to use FD.io VPP with virtual machines. We describe how to create Vhost port with VPP and how to connect them to VPP. We will also discuss some limitations of Vhost.

3.2.1 Prerequisites

For this use case we will assume FD.io VPP is installed. We will also assume the user can create and start basic virtual machines. This use case will use the linux virsh commands. For more information on virsh refer to [virsh man page](#).

The image that we use is based on an Ubuntu cloud image downloaded from: [Ubuntu Cloud Images](#).

All FD.io VPP commands are being run from a su shell.

3.2.2 Topology

In this case we will use 2 systems. One system we will be running standard linux, the other will be running FD.io VPP.

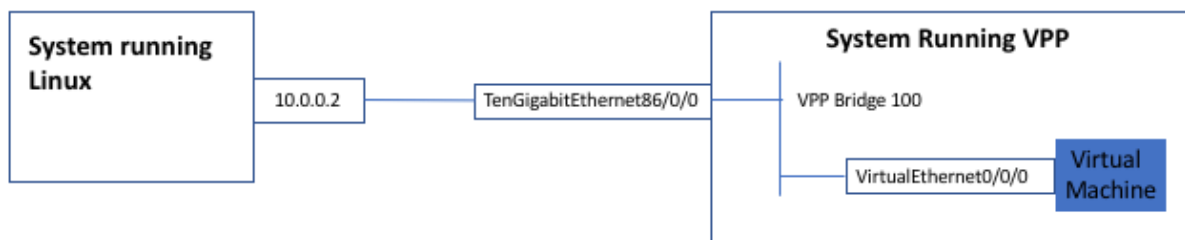


Fig. 1: Vhost Use Case Topology

3.2.3 Creating The Virtual Interface

We will start on the system running FD.io VPP and show that no Virtual interfaces have been created. We do this using the *Show Interface* command.

Notice we do not have any virtual interfaces. We do have an interface (TenGigabitEthernet86/0/0) that is up. This interface is connected to a system running, in our example standard linux. We will use this system to verify our connectivity to our VM with ping.

```
$ sudo bash
# vppctl

  _/ _/ _ \  ( ) _  | | / / _ \ / _ \
 _/ _/ // // / / _ \ | | / / _ \ / _ \
 /_ / / _ _ ( ) _ \ _ / | _ / / / _ /

vpp# clear interfaces
vpp# show int
```

Name	Idx	State	Counter	Count
TenGigabitEthernet86/0/0	1	up		
TenGigabitEthernet86/0/1	2	down		
local0	0	down		

```
vpp#
```

For more information on the interface commands refer to: *Interface Commands*

The next step will be to create the virtual port using the *Create Vhost-User* command. This command will create the virtual port in VPP and create a linux socket that the VM will use to connect to VPP.

The port can be created using VPP as the socket server or client.

Creating the VPP port:

```
vpp# create vhost socket /tmp/vm00.sock
VirtualEthernet0/0/0
vpp# show int
```

Name	Idx	State	Counter	Count
TenGigabitEthernet86/0/0	1	up		
TenGigabitEthernet86/0/1	2	down		
VirtualEthernet0/0/0	3	down		
local0	0	down		

```
vpp#
```

Notice the interface **VirtualEthernet0/0/0**. In this example we created the virtual interface as a client.

We can get more detail on the vhost connection with the *Show Vhost-User* command.

```
vpp# show vhost
Virtio vhost-user interfaces
Global:
  coalesce frames 32 time 1e-3
  number of rx virtqueues in interrupt mode: 0
Interface: VirtualEthernet0/0/0 (ifindex 3)
virtio_net_hdr_sz 12
features mask (0xffffffffffffffff):
features (0x58208000):
  VIRTIO_NET_F_MRG_RXBUF (15)
  VIRTIO_NET_F_GUEST_ANNOUNCE (21)
  VIRTIO_F_ANY_LAYOUT (27)
```

(continues on next page)

(continued from previous page)

```

VIRTIO_F_INDIRECT_DESC (28)
VHOST_USER_F_PROTOCOL_FEATURES (30)
protocol features (0x3)
VHOST_USER_PROTOCOL_F_MQ (0)
VHOST_USER_PROTOCOL_F_LOG_SHMFD (1)

socket filename /tmp/vm00.sock type client errno "No such file or directory"

rx placement:
tx placement: spin-lock
  thread 0 on vring 0
  thread 1 on vring 0

Memory regions (total 0)

```

Notice **No such file or directory** and **Memory regions (total 0)**. This is because the VM has not been created yet.

3.2.4 Creating the Virtual Machine

We will now create the virtual machine. We use the “virsh create command”. For the complete file we use refer to [An XML File](#).

It is important to note that in the XML file we specify the socket path that is used to connect to FD.io VPP.

This is done with a section that looks like this

```

<interface type='vhostuser'>
  <mac address='52:54:00:4c:47:f2' />
  <source type='unix' path='/tmp/vm00.sock' mode='server' />
  <model type='virtio' />
  <alias name='net1' />
  <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0' />
</interface>

```

Notice the **interface type** and the **path** to the socket.

Now we create the VM. The virsh list command shows the VMs that have been created. We start with no VMs.

```

$ virsh list
Id      Name                                     State
-----

```

Create the VM with the virsh create command specifying our xml file.

```

$ virsh create ./iperf3-vm.xml
Domain iperf-server3 created from ./iperf3-vm.xml

$ virsh list
Id      Name                                     State
-----
65      iperf-server3                           running

```

The VM is now created.

Note: After a VM is created an xml file can be created with “virsh dumpxml”.

```
$ virsh dumpxml iperf-server3
<domain type='kvm' id='65'>
  <name>iperf-server3</name>
  <uuid>e23d37c1-10c3-4a6e-ae99-f315a4165641</uuid>
  <memory unit='KiB'>262144</memory>
  .....
```

Once the virtual machine is created notice the socket filename shows **Success** and there are **Memory Regions**. At this point the VM and FD.io VPP are connected. Also notice **qsz 256**. This system is running an older version of qemu. A queue size of 256 will affect vhost throughput. The qsz should be 1024. On the web you should be able to find ways to install a newer version of qemu or change the queue size.

```
vpp# show vhost
Virtio vhost-user interfaces
Global:
  coalesce frames 32 time 1e-3
  number of rx virtqueues in interrupt mode: 0
Interface: VirtualEthernet0/0/0 (ifindex 3)
virtio_net_hdr_sz 12
features mask (0xffffffffffffffff):
features (0x58208000):
  VIRTIO_NET_F_MRG_RXBUF (15)
  VIRTIO_NET_F_GUEST_ANNOUNCE (21)
  VIRTIO_F_ANY_LAYOUT (27)
  VIRTIO_F_INDIRECT_DESC (28)
  VHOST_USER_F_PROTOCOL_FEATURES (30)
protocol features (0x3)
  VHOST_USER_PROTOCOL_F_MQ (0)
  VHOST_USER_PROTOCOL_F_LOG_SHMFD (1)

socket filename /tmp/vm00.sock type client errno "Success"

rx placement:
  thread 1 on vring 1, polling
tx placement: spin-lock
  thread 0 on vring 0
  thread 1 on vring 0

Memory regions (total 2)
region fd    guest_phys_addr    memory_size    userspace_addr    mmap_offset
→ mmap_addr
=====
→=====
0    31    0x0000000000000000 0x00000000000a0000 0x00007f1db9c00000
→0x0000000000000000 0x00007f7db0400    000
1    32    0x00000000000c0000 0x000000000ff40000 0x00007f1db9cc0000
→0x00000000000c0000 0x00007f7d94ec0    000

Virtqueue 0 (TX)
qsz 256 last_avail_idx 0 last_used_idx 0
avail.flags 0 avail.idx 256 used.flags 1 used.idx 0
kickfd 33 callfd 34 errfd -1

Virtqueue 1 (RX)
qsz 256 last_avail_idx 8 last_used_idx 8
avail.flags 0 avail.idx 8 used.flags 1 used.idx 8
kickfd 29 callfd 35 errfd -1
```

3.2.5 Bridge the Interfaces

To connect the 2 interfaces we put them on an L2 bridge.

Use the “set interface l2 bridge” command.

```
vpp# set interface l2 bridge VirtualEthernet0/0/0 100
vpp# set interface l2 bridge TenGigabitEthernet86/0/0 100
vpp# show bridge
  BD-ID   Index   BSN   Age(min)   Learning   U-Forwrd   UU-Flood   Flooding   ARP-Term
↪BVI-Intf
  100     1       0     off        on         on         on         on         off
↪N/A
vpp# show bridge 100 det
  BD-ID   Index   BSN   Age(min)   Learning   U-Forwrd   UU-Flood   Flooding   ARP-Term
↪BVI-Intf
  100     1       0     off        on         on         on         on         off
↪N/A

      Interface                If-idx  ISN   SHG   BVI   TxFlood      VLAN-Tag-Rewrite
VirtualEthernet0/0/0          3       1    0    -     *           none
TenGigabitEthernet86/0/0      1       1    0    -     *           none
vpp# show vhost
```

3.2.6 Bring the Interfaces Up

We can now bring all the pertinent interfaces up. We can then we will then be able to communicate with the VM from the remote system running Linux.

Bring the interfaces up with *Set Interface State* command.

```
vpp# set interface state VirtualEthernet0/0/0 up
vpp# set interface state TenGigabitEthernet86/0/0 up
vpp# sh int
```

Name	Idx	State	Counter	Count
TenGigabitEthernet86/0/0	1	up	rx packets	2
			rx bytes	180
TenGigabitEthernet86/0/1	2	down		
VirtualEthernet0/0/0	3	up	tx packets	2
			tx bytes	180
local0	0	down		

3.2.7 Ping from the VM

The remote Linux system has an ip address of “10.0.0.2” we can now reach it from the VM.

Use the “virsh console” command to attach to the VM. “ctrl-D” to exit.

```
$ virsh console iperf-server3
Connected to domain iperf-server3
Escape character is ^]

Ubuntu 16.04.3 LTS iperfvm ttyS0
.....
```

(continues on next page)

(continued from previous page)

```

root@iperfvm:~# ping 10.0.0.2
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.285 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.154 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.159 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.208 ms

```

On VPP you can now see the packet counts increasing. The packets from the VM are seen as **rx packets** on **VirtualEthernet0/0/0**, they are then bridged to **TenGigabitEthernet86/0/0** and are seen leaving the system as **tx packets**. The reverse is true on the way in.

```

vpp# sh int

```

Name	Idx	State	Counter	Count
TenGigabitEthernet86/0/0	1	up	rx packets	16
			rx bytes	1476
			tx packets	14
			tx bytes	1260
TenGigabitEthernet86/0/1	2	down		
VirtualEthernet0/0/0	3	up	rx packets	14
			rx bytes	1260
			tx packets	16
			tx bytes	1476
local0	0	down		

```

vpp#

```

3.2.8 Cleanup

Destroy the VMs with “virsh destroy”

```

cto@tf-ucs-3:~$ virsh list

```

Id	Name	State
65	iperf-server3	running

```

cto@tf-ucs-3:~$ virsh destroy iperf-server3
Domain iperf-server3 destroyed

```

Delete the Virtual port in FD.io VPP

```

vpp# delete vhost-user VirtualEthernet0/0/0
vpp# show int

```

Name	Idx	State	Counter	Count
TenGigabitEthernet86/0/0	1	up	rx packets	21
			rx bytes	1928
			tx packets	19
			tx bytes	1694
TenGigabitEthernet86/0/1	2	down		
local0	0	down		

Restart FD.io VPP

```

# service vpp restart
# vppctl show int

```

Name	Idx	State	Counter	Count
TenGigabitEthernet86/0/0	1	down		

(continues on next page)

(continued from previous page)

TenGigabitEthernet86/0/1	2	down
local0	0	down

3.2.9 Limitations

There are some limitations when using the qemu vhost driver. Some are described in this section.

Performance

VPP performance with vHost is limited by the Qemu vHost driver. FD.io VPP 18.04 CSIT vHost testing shows with 2 threads, 2 cores and a Queue size of 1024 the maximum NDR throughput was about 7.5 Mpps. This is about the limit at this time.

For all the details on the CSIT VM vhost connection refer to the [CSIT VM vHost performance tests](#).

Features

These are the features not supported with FD.io VPP vHost.

- VPP implements vHost in device mode only. VPP is intended to work with Qemu which implements vHost in driver mode, it does not implement vHost driver mode.
- VPP vHost implementation does not support checksum or transmit segmentation offload.
- VPP vHost implementation does not support packet receive filtering feature for controlling receive traffic.

3.2.10 The XML File

An example of a file that could be used with the virsh create command.

```
<domain type='kvm' id='54'>
  <name>iperf-server</name>
  <memory unit='KiB'>1048576</memory>
  <currentMemory unit='KiB'>1048576</currentMemory>
  <memoryBacking>
    <hugepages>
      <page size='2048' unit='KiB' />
    </hugepages>
  </memoryBacking>
  <vcpu placement='static'>1</vcpu>
  <resource>
    <partition>/machine</partition>
  </resource>
  <os>
    <type arch='x86_64' machine='pc-i440fx-xenial'>hvm</type>
    <boot dev='hd' />
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <cpu mode='host-model'>
```

(continues on next page)

(continued from previous page)

```

<model fallback='allow'></model>
<numa>
  <cell id='0' cpus='0' memory='262144' unit='KiB' memAccess='shared' />
</numa>
</cpu>
<clock offset='utc'>
  <timer name='rtc' tickpolicy='catchup' />
  <timer name='pit' tickpolicy='delay' />
  <timer name='hpet' present='no' />
</clock>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<pm>
  <suspend-to-mem enabled='no' />
  <suspend-to-disk enabled='no' />
</pm>
<devices>
  <emulator>/usr/bin/kvm</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2' />
    <source file='/tmp/xenial-mod.img' />
    <backingStore />
    <target dev='vda' bus='virtio' />
    <alias name='virtio-disk0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0' />
  </disk>
  <disk type='file' device='cdrom'>
    <driver name='qemu' type='raw' />
    <source file='/scratch/jdenisco/sae/configs/cloud-config.iso' />
    <backingStore />
    <target dev='hda' bus='ide' />
    <readonly />
    <alias name='ide0-0-0' />
    <address type='drive' controller='0' bus='0' target='0' unit='0' />
  </disk>
  <controller type='usb' index='0' model='ich9-ehci1'>
    <alias name='usb' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x7' />
  </controller>
  <controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0' />
  </controller>
  <controller type='ide' index='0'>
    <alias name='ide' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1' />
  </controller>
  <controller type='virtio-serial' index='0'>
    <alias name='virtio-serial0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0' />
  </controller>
  <interface type='vhostuser'>
    <mac address='52:54:00:4c:47:f2' />
    <source type='unix' path='/tmp/vm00.sock' mode='server' />
    <model type='virtio' />
    <alias name='net1' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0' />

```

(continues on next page)

(continued from previous page)

```

</interface>
<serial type='pty'>
  <source path='/dev/pts/2' />
  <target port='0' />
  <alias name='serial0' />
</serial>
<console type='pty' tty='/dev/pts/2'>
  <source path='/dev/pts/2' />
  <target type='serial' port='0' />
  <alias name='serial0' />
</console>
<input type='mouse' bus='ps2' />
<input type='keyboard' bus='ps2' />
<graphics type='vnc' port='5900' autoport='yes' listen='127.0.0.1'>
  <listen type='address' address='127.0.0.1' />
</graphics>
<memballoon model='virtio'>
  <alias name='balloon0' />
  <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0' />
</memballoon>
</devices>
<seclabel type='dynamic' model='apparmor' relabel='yes'>
  <label>libvirt-2c4c9317-c7a5-4b37-b789-386ccda7348a</label>
  <imagelabel>libvirt-2c4c9317-c7a5-4b37-b789-386ccda7348a</imagelabel>
</seclabel>
</domain>

```

3.3 Using VPP as a Home Gateway

Vpp running on a small system (with appropriate NICs) makes a fine home gateway. The resulting system performs far in excess of requirements: a TAG=vpp_debug image runs at a vector size of ~1.1 terminating a 90-mbit down / 10-mbit up cable modem connection.

At a minimum, install sshd and the isc-dhcp-server. If you prefer, you can use dnsmasq.

3.3.1 Configuration files

/etc/vpp/startup.conf:

```

unix {
  nodaemon
  log /var/log/vpp/vpp.log
  full-coredump
  cli-listen /run/vpp/cli.sock
  startup-config /setup.gate
  gid vpp
}
api-segment {
  gid vpp
}
dpdk {
  dev 0000:03:00.0
  dev 0000:14:00.0

```

(continues on next page)

(continued from previous page)

```

    etc.
    poll-sleep 10
}

```

isc-dhcp-server configuration:

```

subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.10 192.168.1.99;
    option routers 192.168.1.1;
    option domain-name-servers 8.8.8.8;
}

```

If you decide to enable the vpp dns name resolver, substitute 192.168.1.2 for 8.8.8.8 in the dhcp server configuration.

/etc/ssh/sshd_config:

```

# What ports, IPs and protocols we listen for
Port <REDACTED-high-number-port>
# Change to no to disable tunnelled clear text passwords
PasswordAuthentication no

```

For your own comfort and safety, do NOT allow password authentication and do not answer ssh requests on port 22. Experience shows several hack attempts per hour on port 22, but none (ever) on random high-number ports.

vpp configuration:

```

comment { This is the WAN interface }
set int state GigabitEthernet3/0/0 up
comment { set int mac address GigabitEthernet3/0/0 mac-to-clone-if-needed }
set dhcp client intfc GigabitEthernet3/0/0 hostname vppgate

comment { Create a BVI loopback interface}
loop create
set int 12 bridge loop0 1 bvi
set int ip address loop0 192.168.1.1/24
set int state loop0 up

comment { Add more inside interfaces as needed ... }
set int 12 bridge GigabitEthernet0/14/0 1
set int state GigabitEthernet0/14/0 up

comment { dhcp server and host-stack access }
tap connect lstack address 192.168.1.2/24
set int 12 bridge tapcli-0 1
set int state tapcli-0 up

comment { Configure NAT}
nat44 add interface address GigabitEthernet3/0/0
set interface nat44 in loop0 out GigabitEthernet3/0/0

comment { allow inbound ssh to the <REDACTED-high-number-port> }
nat44 add static mapping local 192.168.1.2 <REDACTED> external GigabitEthernet3/0/0
-><REDACTED> tcp

comment { if you want to use the vpp DNS server, add the following }
comment { Remember to adjust the isc-dhcp-server configuration appropriately }
comment { nat44 add identity mapping external GigabitEthernet3/0/0 udp 53053 }

```

(continues on next page)

(continued from previous page)

```
comment { bin dns_name_server_add_del 8.8.8.8 }
comment { bin dns_name_server_add_del 68.87.74.166 }
comment { bin dns_enable_disable }
comment { see patch below, which adds these commands }
service restart isc-dhcp-server
add default linux route via 192.168.1.1
```

3.3.2 Patches

You'll need this patch to add the “service restart” and “add default linux route” commands:

```
diff --git a/src/vpp/vnet/main.c b/src/vpp/vnet/main.c
index 6e136e19..69189c93 100644
--- a/src/vpp/vnet/main.c
+++ b/src/vpp/vnet/main.c
@@ -18,6 +18,8 @@
#include <vlib/unix/unix.h>
#include <vnet/plugin/plugin.h>
#include <vnet/ethernet/ethernet.h>
+#include <vnet/ip/ip4_packet.h>
+#include <vnet/ip/format.h>
#include <vpp/app/version.h>
#include <vpp/api/vpe_msg_enum.h>
#include <limits.h>
@@ -400,6 +402,63 @@ VLIB_CLI_COMMAND (test_crash_command, static) = {

#ifdef

+static clib_error_t *
+restart_isc_dhcp_server_command_fn (vlib_main_t * vm,
+                                unformat_input_t * input,
+                                vlib_cli_command_t * cmd)
+{
+  int rv __attribute__((unused));
+  /* Wait three seconds... */
+  vlib_process_suspend (vm, 3.0);
+  rv = system ("/usr/sbin/service isc-dhcp-server restart");
+  vlib_cli_output (vm, "Restarted the isc-dhcp-server...");
+  return 0;
+}
+
+/* *INDENT-OFF* */
+VLIB_CLI_COMMAND (restart_isc_dhcp_server_command, static) = {
+  .path = "service restart isc-dhcp-server",
+  .short_help = "restarts the isc-dhcp-server",
+  .function = restart_isc_dhcp_server_command_fn,
+};
+/* *INDENT-ON* */
+
+static clib_error_t *
+add_default_linux_route_command_fn (vlib_main_t * vm,
+                                unformat_input_t * input,
+                                vlib_cli_command_t * c)
```

(continues on next page)

(continued from previous page)

```

+{
+  int rv __attribute__((unused));
+  ip4_address_t ip4_addr;
+  u8 *cmd;
+
+  if (!unformat (input, "%U", unformat_ip4_address, &ip4_addr))
+    return clib_error_return (0, "default gateway address required...");
+
+  cmd = format (0, "/sbin/route add -net 0.0.0.0/0 gw %U",
+               format_ip4_address, &ip4_addr);
+  vec_add1 (cmd, 0);
+
+  rv = system (cmd);
+
+  vlib_cli_output (vm, "%s", cmd);
+
+  vec_free (cmd);
+
+  return 0;
+}
+
+/* *INDENT-OFF* */
+VLIB_CLI_COMMAND (add_default_linux_route_command, static) = {
+  .path = "add default linux route via",
+  .short_help = "Adds default linux route: 0.0.0.0/0 via <addr>",
+  .function = add_default_linux_route_command_fn,
+};
+/* *INDENT-ON* */
+
+
+

```

3.3.3 Using the temporal mac filter plugin

If you need to restrict network access for certain devices to specific daily time ranges, configure the “mactime” plugin. Enable the feature on the NAT “inside” interfaces:

```

bin mactime_enable_disable GigabitEthernet0/14/0
bin mactime_enable_disable GigabitEthernet0/14/1
...

```

Create the required src-mac-address rule database. There are 4 rule entry types:

- allow-static - pass traffic from this mac address
- drop-static - drop traffic from this mac address
- allow-range - pass traffic from this mac address at specific times
- drop-range - drop traffic from this mac address at specific times

Here are some examples:

```

bin mactime_add_del_range name alarm-system mac 00:de:ad:be:ef:00 allow-static
bin mactime_add_del_range name unwelcome mac 00:de:ad:be:ef:01 drop-static
bin mactime_add_del_range name not-during-business-hours mac <mac> drop-range Mon -
↪Fri 7:59 - 18:01
bin mactime_add_del_range name monday-busines-hours mac <mac> allow-range Mon 7:59 -
↪18:01

```

(continues on next page)

(continued from previous page)

3.4 vSwitch/vRouter

3.4.1 FD.io VPP as a vSwitch/vRouter

Note: We need to provide commands and and show how to use VPP as a vSwitch/vRouter

One of the use cases for the FD.io 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 FD.io VPP platform. For more in depth descriptions about other possible use cases, see the list of

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

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)

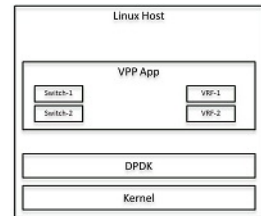


Fig. 2: Figure: Linux host as a vSwitch

CHAPTER 4

Troubleshooting

This chapter describes some of the many techniques used to troubleshoot and diagnose problem with FD.io VPP implementations.

4.1 How to Report an Issue

4.1.1 Reporting Bugs

Although every situation is different, this page describes how to collect data which will help make efficient use of everyone's time when dealing with vpp bugs.

Before you press the Jira button to create a bug report - or email vpp-dev@lists.fd.io - please ask yourself whether there's enough information for someone else to understand and possibly to reproduce the issue given a reasonable amount of effort. **Unicast emails to maintainers, committers, and the project PTL are strongly discouraged.**

A good strategy for clear-cut bugs: file a detailed Jira ticket, and then send a short description of the issue to vpp-dev@lists.fd.io, perhaps from the Jira ticket description. It's fine to send email to vpp-dev@lists.fd.io to ask a few questions **before** filing Jira tickets.

4.1.2 Data to include in bug reports

Image version and operating environment

Please make sure to include the vpp image version and command-line arguments.

```
$ sudo bash
# vppctl show version verbose cmdline
Version:                v18.07-rc0~509-gb9124828
Compiled by:             vppuser
Compile host:            vppbuild
Compile date:            Fri Jul 13 09:05:37 EDT 2018
```

(continues on next page)

(continued from previous page)

```
Compile location:      /scratch/vpp-showversion
Compiler:             GCC 7.3.0
Current PID:          5211
Command line arguments:
  /scratch/vpp-showversion/build-root/install-vpp_debug-native/vpp/bin/vpp
  unix
  interactive
```

With respect to the operating environment: if misbehavior involving a specific VM / container / bare-metal environment is involved, please describe the environment in detail:

- Linux Distro (e.g. Ubuntu 14.04.3 LTS, CentOS-7, etc.)
- NIC type(s) (ixgbe, i40e, enic, etc. etc.), vhost-user, tuntap
- NUMA configuration if applicable

Please note the CPU architecture (x86_86, aarch64), and hardware platform.

When practicable, please report issues against released software, or unmodified master/latest software.

“Show” command output

Every situation is different. If the issue involves a sequence of debug CLI command, please enable CLI command logging, and send the sequence involved. Note that the debug CLI is a developer’s tool - **no warranty express or implied** - and that we may choose not to fix debug CLI bugs.

Please include “show error” [error counter] output. It’s often helpful to “clear error”, send a bit of traffic, then “show error” particularly when running vpp on a noisy networks.

Please include ip4 / ip6 / mpls FIB contents (“show ip fib”, “show ip6 fib”, “show mpls fib”, “show mpls tunnel”).

Please include “show hardware”, “show interface”, and “show interface address” output

Here is a consolidated set of commands that are generally useful before/after sending traffic. Before sending traffic.

```
vppctl clear hardware
vppctl clear interface
vppctl clear error
vppctl clear run
```

Send some traffic and then issue the following commands.

```
vppctl show version verbose
vppctl show hardware
vppctl show hardware address
vppctl show interface
vppctl show run
vppctl show error
```

Here are some protocol specific show commands that may also make sense. Only include those features which have been configured.

```
vppctl show l2fib
vppctl show bridge-domain

vppctl show ip fib
vppctl show ip arp
```

(continues on next page)

(continued from previous page)

```
vppctl show ip6 fib
vppctl show ip6 neighbors

vppctl show mpls fib
vppctl show mpls tunnel
```

Network Topology

Please include a crisp description of the network topology, including L2 / IP / MPLS / segment-routing addressing details. If you expect folks to reproduce and debug issues, this is a must.

At or above a certain level of topological complexity, it becomes problematic to reproduce the original setup.

Packet Tracer Output

If you capture packet tracer output which seems relevant, please include it.

```
vppctl trace add dpdk-input 100 # or similar
```

send-traffic

```
vppctl show trace
```

4.1.3 Capturing post-mortem data

It should go without saying, but anyhow: **please put post-mortem data in obvious, accessible places.** Time wasted trying to acquire accounts, credentials, and IP addresses simply delays problem resolution.

Please remember to add post-mortem data location information to Jira tickets.

Syslog Output

The vpp signal handler typically writes a certain amount of data in /var/log/syslog before exiting. Make sure to check for evidence, e.g via “grep /usr/bin/vpp /var/log/syslog” or similar.

Binary API Trace

If the issue involves a sequence of control-plane API messages - even a very long sequence - please enable control-plane API tracing. Control-plane API post-mortem traces end up in /tmp/api_post_mortem.<pid>.

Please remember to put post-mortem binary api traces in accessible places.

These API traces are especially helpful in cases where the vpp engine is throwing traffic on the floor, e.g. for want of a default route or similar.

Make sure to leave the default stanza “... api-trace { on } ... ” in the vpp startup configuration file /etc/vpp/startup.conf, or to include it in the command line arguments passed by orchestration software.

Core Files

Production systems, as well as long-running pre-production soak-test systems, **must** arrange to collect core images. There are various ways to configure core image capture, including e.g. the Ubuntu “corekeeper” package. In a pinch, the following very basic sequence will capture usable vpp core files in /tmp/dumps.

```
# mkdir -p /tmp/dumps
# sysctl -w debug.exception-trace=1
# sysctl -w kernel.core_pattern="/tmp/dumps/%e-%t"
# ulimit -c unlimited
# echo 2 > /proc/sys/fs/suid_dumpable
```

Vpp core files often appear enormous. Gzip typically compresses them to manageable sizes. A multi-GByte corefile often compresses to 10-20 Mbytes.

Please remember to put compressed core files in accessible places.

Make sure to leave the default stanza “... unix { ... full-coredump ... } ... ” in the vpp startup configuration file /etc/vpp/startup.conf, or to include it in the command line arguments passed by orchestration software.

Core files from private, modified images are discouraged. If it’s necessary to go that route, please copy the **exact** Debian packages (or RPMs) corresponding to the core file to the same public place as the core file. In particular.

- vpp_<version>_<arch>.deb # the vpp executable
- vpp-dbg_<version>_<arch>.deb # debug symbols
- vpp-dev_<version>_<arch>.deb # development package
- vpp-lib_<version>_<arch>.deb # shared libraries
- vpp-plugins_<version>_<arch>.deb # plugins

Please include the full commit-ID the Jira ticket.

If we go through the setup process only to discover that the image and core files don’t match, it will simply delay resolution of the issue. And it will annoy the heck out of the engineer who just wasted their time. Exact means **exact**, not “oh, gee, I added a few lines of debug scaffolding since then...”

4.2 CPU Load/Usage

There are various commands and tools that can help users see FD.io VPP CPU and memory usage at runtime.

4.2.1 Linux top/htop

The Linux top and htop are decent tools to look at FD.io VPP cpu and memory usage, but they will only show preallocated memory and total CPU usage. These commands can be useful to show which cores VPP is running on.

This is an example of VPP instance that is running on cores 8 and 9. For this output type **top** and then type **1** when the tool starts.

```
$ top
top - 11:04:04 up 35 days,  3:16,  5 users,  load average: 2.33, 2.23, 2.16
Tasks: 435 total,   2 running, 432 sleeping,   1 stopped,   0 zombie
%Cpu0  :  1.0 us,   0.7 sy,   0.0 ni, 98.0 id,   0.0 wa,   0.0 hi,   0.3 si,   0.0 st
%Cpu1  :  2.0 us,   0.3 sy,   0.0 ni, 97.7 id,   0.0 wa,   0.0 hi,   0.0 si,   0.0 st
```

(continues on next page)

(continued from previous page)

```
%Cpu2 : 0.7 us, 1.0 sy, 0.0 ni, 98.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu3 : 1.7 us, 0.7 sy, 0.0 ni, 97.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu4 : 2.0 us, 0.7 sy, 0.0 ni, 97.4 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu5 : 3.0 us, 0.3 sy, 0.0 ni, 96.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu6 : 2.3 us, 0.7 sy, 0.0 ni, 97.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu7 : 2.6 us, 0.3 sy, 0.0 ni, 97.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu8 : 96.0 us, 0.3 sy, 0.0 ni, 3.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu9 : 100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu10 : 1.0 us, 0.3 sy, 0.0 ni, 98.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
....
```

4.2.2 VPP Memory Usage

For details on VPP memory usage you can use the **show memory** command

This is the example VPP memory usage on 2 cores.

```
# vppctl show memory verbose
Thread 0 vpp_main
22043 objects, 17878k of 20826k used, 2426k free, 2396k reclaimed, 346k overhead,
↪1048572k capacity
  alloc. from small object cache: 22875 hits 39973 attempts (57.23%) replacements 5143
  alloc. from free-list: 44732 attempts, 26017 hits (58.16%), 528461 considered (per-
↪attempt 11.81)
  alloc. from vector-expand: 3430
  allocs: 52324 2027.84 clocks/call
  frees: 30280 594.38 clocks/call
Thread 1 vpp_wk_0
22043 objects, 17878k of 20826k used, 2427k free, 2396k reclaimed, 346k overhead,
↪1048572k capacity
  alloc. from small object cache: 22881 hits 39984 attempts (57.23%) replacements 5148
  alloc. from free-list: 44736 attempts, 26021 hits (58.17%), 528465 considered (per-
↪attempt 11.81)
  alloc. from vector-expand: 3430
  allocs: 52335 2027.54 clocks/call
  frees: 30291 594.36 clocks/call
```

4.2.3 VPP CPU Load

To find the VPP CPU load or how busy VPP is use the **show runtime** command.

With at least one interface in polling mode, the VPP CPU utilization is always 100%.

A good indicator of CPU load is “**average vectors/node**”. A bigger number means VPP is more busy but also more efficient. The Maximum value is 255 (unless you change `VLIB_FRAME_SIZE` in code). It basically means how many packets are processed in batch.

If VPP is not loaded it will likely poll so fast that it will just get one or few packets from the rx queue. This is the case shown below on Thread 1. As load goes up vpp will have more work to do, so it will poll less frequently, and that will result in more packets waiting in rx queue. More packets will result in more efficient execution of the code so number of clock cycles / packet will go down. When “average vectors/node” goes up close to 255, you will likely start observing rx queue tail drops.

```
# vppctl show run
Thread 0 vpp_main (lcore 8)
Time 6152.9, average vectors/node 0.00, last 128 main loops 0.00 per node 0.00
  vector rates in 0.0000e0, out 0.0000e0, drop 0.0000e0, punt 0.0000e0
      Name                State      Calls      Vectors
↳Suspends      Clocks      Vectors/Call
acl-plugin-fa-cleaner-process  event wait      0          0
↳ 1          3.66e4      0.00
admin-up-down-process      event wait      0          0
↳ 1          2.54e3      0.00
....
-----
Thread 1 vpp_wk_0 (lcore 9)
Time 6152.9, average vectors/node 1.00, last 128 main loops 0.00 per node 0.00
  vector rates in 1.3073e2, out 1.3073e2, drop 6.5009e-4, punt 0.0000e0
      Name                State      Calls      Vectors
↳Suspends      Clocks      Vectors/Call
TenGigabitEthernet86/0/0-outpu  active      804395      804395
↳ 0          6.17e2      1.00
TenGigabitEthernet86/0/0-tx      active      804395      804395
↳ 0          7.29e2      1.00
arp-input      active          2          2
↳ 0          3.82e4      1.00
dpdk-input      polling      24239296364  804398
↳ 0          1.59e7      0.00
error-drop      active          4          4
↳ 0          4.65e3      1.00
ethernet-input  active          2          2
↳ 0          1.08e4      1.00
interface-output      active          1          1
↳ 0          3.78e3      1.00
ip4-glean      active          1          1
↳ 0          6.98e4      1.00
ip4-icmp-echo-request      active      804394      804394
↳ 0          5.02e2      1.00
ip4-icmp-input      active      804394      804394
↳ 0          4.63e2      1.00
ip4-input-no-checksum      active      804394      804394
↳ 0          8.51e2      1.00
ip4-load-balance      active      804394      804394
↳ 0          5.46e2      1.00
ip4-local      active      804394      804394
↳ 0          5.79e2      1.00
ip4-lookup      active      804394      804394
↳ 0          5.71e2      1.00
ip4-rewrite      active      804393      804393
↳ 0          5.69e2      1.00
ip6-input      active          2          2
↳ 0          5.72e3      1.00
ip6-not-enabled      active          2          2
↳ 0          1.56e4      1.00
unix-epoll-input      polling      835722      0
↳ 0          3.03e-3      0.00
```


5.1 Progressive VPP Tutorial

5.1.1 Overview

Learn to run FD.io VPP on a single Ubuntu 16.04 VM using Vagrant with this walkthrough covering basic FD.io VPP scenarios. Useful FD.io VPP commands will be used, and will discuss basic operations, and the state of a running FD.io VPP on a system.

Note: This is *not* intended to be a ‘How to Run in a Production Environment’ set of instructions.

5.1.2 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 or root access, 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 in the steps below.

5.1.3 Running Vagrant

FD.io 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 FD.io 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, FD.io VPP supports this.

When running multiple FD.io 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.

Setting up FD.io VPP environment with Vagrant

Refer to *this guide* for setting up a Virtual Box with Vagrant

After setting up Vagrant, use these commands on your Vagrant directory to boot the VM:

```
# vagrant up
```

```
# vagrant ssh
```

Afterwards, configure FD.io VPP on the Vagrant system following the steps on the *VPP Configuration Utility* guide.

The DPDK Plugin will be disabled for this section. The link below demonstrates how this is done.

Removing the DPDK Plugin

For the purposes of this tutorial, the `dpdk` plugin will be removed. To do this edit the `startup.conf` file with the following, your `startup.conf` file may already have this line commented, and may just need to uncomment it:

```
plugins
{
    plugin dpdk_plugin.so { disable }
}
```

5.1.4 Start a FD.io VPP shell using vppctl

The command `$ sudo vppctl` will launch a FD.io VPP shell with which you can run multiple FD.io VPP commands interactively by running:

```
$ sudo vppctl  
_/_/_/_/_\ (_)_ _ | | /_/_- \/_\  
_/_// // / / / - \ | | / _/_/_/  
/_/_/_(_)-\_/_/_|_|/_/_/_/_/_  
vpp# show ver  
vpp v18.07-release built by root on c469eba2a593 at Mon Jul 30 23:27:03 UTC 2018
```

5.1.5 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

Interface

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

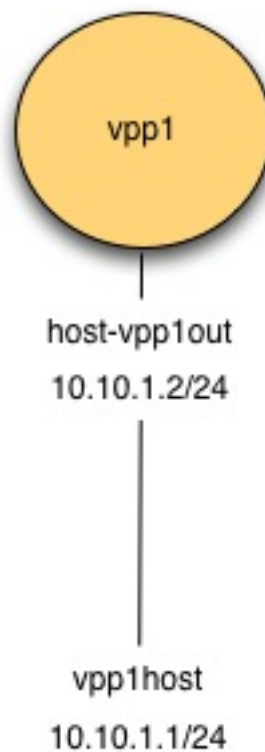


Fig. 1: Figure: Create Interface Topology

Initial State

The initial state here is presumed to be the final state from the exercise [VPP Basics](#)

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 **vpplout** and the other named **vpplhost**

```
$ sudo ip link add name vpplout type veth peer name vpplhost
```

Turn up both ends:

```
$ sudo ip link set dev vpplout up
$ sudo ip link set dev vpplhost up
```

Assign an IP address

```
$ sudo ip addr add 10.10.1.1/24 dev vpplhost
```

Display the result:

```
$ sudo ip addr show vpplhost
5: vpplhost@vpplout: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state_  
↪UP group default qlen 1000  
    link/ether e2:0f:1e:59:ec:f7 brd ff:ff:ff:ff:ff:ff  
    inet 10.10.1.1/24 scope global vpplhost  
        valid_lft forever preferred_lft forever  
    inet6 fe80::e00f:1eff:fe59:ecf7/64 scope link  
        valid_lft forever preferred_lft forever
```

Create vpp host-interface

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

```
vpp# create host-interface name vpplout  
host-vpplout
```

Confirm the interface:

```
vpp# show hardware
```

Name	Idx	Link	Hardware
host-vpplout	1	up	host-vpplout
Ethernet address 02:fe:d9:75:d5:b4			
Linux PACKET socket interface			
local0	0	down	local0
local			

Turn up the interface:

```
vpp# set int state host-vpplout up
```

Confirm the interface is up:

```
vpp# show int
```

	Name	Idx	State	MTU (L3/IP4/IP6/MPLS)	Counter
↔	Count				
	host-vpplout	1	up	9000/0/0/0	
	local0	0	down	0/0/0/0	

Assign ip address 10.10.1.2/24

```
vpp# set int ip address host-vpplout 10.10.1.2/24
```

Confirm the ip address is assigned:

```
vpp# show int addr
host-vpplout (up):
  L3 10.10.1.2/24
local0 (dn):
```

5.1.6 Traces

Skills to be Learned

1. Setup a 'trace'
2. View a 'trace'
3. Clear a 'trace'
4. Verify using ping from host
5. Ping from vpp
6. Examine Arp Table
7. Examine ip fib

Traces

Basic Trace Commands

Show trace buffer [max COUNT].

```
vpp# show trace
```

Clear trace buffer and free memory.

```
vpp# clear trace
```

filter trace output - include NODE COUNT | exclude NODE COUNT | none.

```
vpp# trace filter <include NODE COUNT | exclude NODE COUNT | none>
```

Skills to be Learned

1. Setup a 'trace'
2. View a 'trace'
3. Clear a 'trace'
4. Verify using ping from host
5. Ping from vpp
6. Examine Arp Table
7. Examine ip fib

Add Trace

```
vpp# trace add af-packet-input 10
```

Ping from Host to FD.io VPP

```
vpp# q
$ 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=0.283 ms

--- 10.10.1.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.283/0.283/0.283/0.000 ms
```

Examine Trace of ping from host to FD.io VPP

```
# vppctl
vpp# show trace
----- Start of thread 0 vpp_main -----
Packet 1

00:17:04:099260: 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 0x5b60e370 nsec 0x3af2736f vlan 0 vlan_tpid 0
00:17:04:099269: ethernet-input
IP4: e2:0f:1e:59:ec:f7 -> 02:fe:d9:75:d5:b4
00:17:04:099285: ip4-input
ICMP: 10.10.1.1 -> 10.10.1.2
  tos 0x00, ttl 64, length 84, checksum 0x3f7c
  fragment id 0xe516, flags DONT_FRAGMENT
ICMP echo_request checksum 0xc043
00:17:04:099290: ip4-lookup
fib 0 dpo-idx 5 flow hash: 0x00000000
ICMP: 10.10.1.1 -> 10.10.1.2
```

(continues on next page)

(continued from previous page)

```

    tos 0x00, ttl 64, length 84, checksum 0x3f7c
    fragment id 0xe516, flags DONT_FRAGMENT
ICMP echo_request checksum 0xc043
00:17:04:099296: ip4-local
    ICMP: 10.10.1.1 -> 10.10.1.2
        tos 0x00, ttl 64, length 84, checksum 0x3f7c
        fragment id 0xe516, flags DONT_FRAGMENT
        ICMP echo_request checksum 0xc043
00:17:04:099300: ip4-icmp-input
ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3f7c
    fragment id 0xe516, flags DONT_FRAGMENT
ICMP echo_request checksum 0xc043
00:17:04:099301: ip4-icmp-echo-request
ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3f7c
    fragment id 0xe516, flags DONT_FRAGMENT
ICMP echo_request checksum 0xc043
00:17:04:099303: ip4-load-balance
fib 0 dpo-idx 13 flow hash: 0x00000000
ICMP: 10.10.1.2 -> 10.10.1.1
    tos 0x00, ttl 64, length 84, checksum 0x4437
    fragment id 0xe05b, flags DONT_FRAGMENT
ICMP echo_reply checksum 0xc843
00:17:04:099305: ip4-rewrite
tx_sw_if_index 1 dpo-idx 1 : ipv4 via 10.10.1.1 host-vpplout: mtu:9000,
→e20f1e59ecf702fed975d5b40800 flow hash: 0x00000000
00000000: e20f1e59ecf702fed975d5b4080045000054e05b4000400144370a0a01020a0a
00000020: 01010000c8437c92000170e3605b000000001c170f000000000001011
00:17:04:099307: host-vpplout-output
host-vpplout
IP4: 02:fe:d9:75:d5:b4 -> e2:0f:1e:59:ec:f7
ICMP: 10.10.1.2 -> 10.10.1.1
    tos 0x00, ttl 64, length 84, checksum 0x4437
    fragment id 0xe05b, flags DONT_FRAGMENT
ICMP echo_reply checksum 0xc843

```

Clear trace buffer

```
vpp# clear trace
```

Ping from FD.io VPP to Host

```

vpp# ping 10.10.1.1
64 bytes from 10.10.1.1: icmp_seq=1 ttl=64 time=.0789 ms
64 bytes from 10.10.1.1: icmp_seq=2 ttl=64 time=.0619 ms
64 bytes from 10.10.1.1: icmp_seq=3 ttl=64 time=.0519 ms
64 bytes from 10.10.1.1: icmp_seq=4 ttl=64 time=.0514 ms
64 bytes from 10.10.1.1: icmp_seq=5 ttl=64 time=.0526 ms

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

```

Examine Trace of ping from FD.io VPP to host

The output will demonstrate FD.io VPP's trace of ping for all packets.

```
vpp# show trace
----- Start of thread 0 vpp_main -----
Packet 1

00:17:04:099260: 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 0x5b60e370 nsec 0x3af2736f vlan 0 vlan_tpid 0
00:17:04:099269: ethernet-input
IP4: e2:0f:1e:59:ec:f7 -> 02:fe:d9:75:d5:b4
00:17:04:099285: ip4-input
ICMP: 10.10.1.1 -> 10.10.1.2
  tos 0x00, ttl 64, length 84, checksum 0x3f7c
  fragment id 0xe516, flags DONT_FRAGMENT
ICMP echo_request checksum 0xc043
00:17:04:099290: 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 0x3f7c
  fragment id 0xe516, flags DONT_FRAGMENT
ICMP echo_request checksum 0xc043
00:17:04:099296: ip4-local
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3f7c
    fragment id 0xe516, flags DONT_FRAGMENT
  ICMP echo_request checksum 0xc043
00:17:04:099300: ip4-icmp-input
ICMP: 10.10.1.1 -> 10.10.1.2
  tos 0x00, ttl 64, length 84, checksum 0x3f7c
  fragment id 0xe516, flags DONT_FRAGMENT
ICMP echo_request checksum 0xc043
00:17:04:099301: ip4-icmp-echo-request
ICMP: 10.10.1.1 -> 10.10.1.2
  tos 0x00, ttl 64, length 84, checksum 0x3f7c
  fragment id 0xe516, flags DONT_FRAGMENT
ICMP echo_request checksum 0xc043
00:17:04:099303: ip4-load-balance
fib 0 dpo-idx 13 flow hash: 0x00000000
ICMP: 10.10.1.2 -> 10.10.1.1
  tos 0x00, ttl 64, length 84, checksum 0x4437
  fragment id 0xe05b, flags DONT_FRAGMENT
ICMP echo_reply checksum 0xc843
00:17:04:099305: ip4-rewrite
tx_sw_if_index 1 dpo-idx 1 : ipv4 via 10.10.1.1 host-vpp1out: mtu:9000_
↪e20f1e59ecf702fed975d5b40800 flow hash: 0x00000000
00000000: e20f1e59ecf702fed975d5b4080045000054e05b4000400144370a0a01020a0a
00000020: 01010000c8437c92000170e3605b000000001c170f000000000001011
00:17:04:099307: host-vpp1out-output
host-vpp1out
IP4: 02:fe:d9:75:d5:b4 -> e2:0f:1e:59:ec:f7
ICMP: 10.10.1.2 -> 10.10.1.1
  tos 0x00, ttl 64, length 84, checksum 0x4437
```

(continues on next page)

(continued from previous page)

```

    fragment id 0xe05b, flags DONT_FRAGMENT
ICMP echo_reply checksum 0xc843

Packet 2

00:17:09:113964: 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 0x5b60e375 nsec 0x3b3bd57d vlan 0 vlan_tpid 0
00:17:09:113974: ethernet-input
ARP: e2:0f:1e:59:ec:f7 -> 02:fe:d9:75:d5:b4
00:17:09:113986: arp-input
request, type ethernet/IP4, address size 6/4
e2:0f:1e:59:ec:f7/10.10.1.1 -> 00:00:00:00:00:00/10.10.1.2
00:17:09:114003: host-vpplout-output
host-vpplout
ARP: 02:fe:d9:75:d5:b4 -> e2:0f:1e:59:ec:f7
reply, type ethernet/IP4, address size 6/4
02:fe:d9:75:d5:b4/10.10.1.2 -> e2:0f:1e:59:ec:f7/10.10.1.1

Packet 3

00:18:16:407079: 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 0x5b60e3b9 nsec 0x90b7566 vlan 0 vlan_tpid 0
00:18:16:407085: ethernet-input
IP4: e2:0f:1e:59:ec:f7 -> 02:fe:d9:75:d5:b4
00:18:16:407090: ip4-input
ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3fe8
    fragment id 0x24ab
ICMP echo_reply checksum 0x37eb
00:18:16:407094: 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 0x3fe8
    fragment id 0x24ab
ICMP echo_reply checksum 0x37eb
00:18:16:407097: ip4-local
    ICMP: 10.10.1.1 -> 10.10.1.2
        tos 0x00, ttl 64, length 84, checksum 0x3fe8
        fragment id 0x24ab
    ICMP echo_reply checksum 0x37eb
00:18:16:407101: ip4-icmp-input
ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3fe8
    fragment id 0x24ab
ICMP echo_reply checksum 0x37eb
00:18:16:407104: ip4-icmp-echo-reply
ICMP echo id 7531 seq 1
00:18:16:407108: ip4-drop
    ICMP: 10.10.1.1 -> 10.10.1.2
        tos 0x00, ttl 64, length 84, checksum 0x3fe8
        fragment id 0x24ab

```

(continues on next page)

(continued from previous page)

```

    ICMP echo_reply checksum 0x37eb
00:18:16:407111: error-drop
ip4-icmp-input: unknown type

Packet 4

00:18:17:409084: 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 0x5b60e3ba nsec 0x90b539f vlan 0 vlan_tpid 0
00:18:17:409088: ethernet-input
IP4: e2:0f:1e:59:ec:f7 -> 02:fe:d9:75:d5:b4
00:18:17:409092: ip4-input
ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3f40
    fragment id 0x2553
ICMP echo_reply checksum 0xcc6d
00:18:17:409095: 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 0x3f40
    fragment id 0x2553
ICMP echo_reply checksum 0xcc6d
00:18:17:409097: ip4-local
    ICMP: 10.10.1.1 -> 10.10.1.2
        tos 0x00, ttl 64, length 84, checksum 0x3f40
        fragment id 0x2553
    ICMP echo_reply checksum 0xcc6d
00:18:17:409099: ip4-icmp-input
ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3f40
    fragment id 0x2553
ICMP echo_reply checksum 0xcc6d
00:18:17:409101: ip4-icmp-echo-reply
ICMP echo id 7531 seq 2
00:18:17:409104: ip4-drop
    ICMP: 10.10.1.1 -> 10.10.1.2
        tos 0x00, ttl 64, length 84, checksum 0x3f40
        fragment id 0x2553
    ICMP echo_reply checksum 0xcc6d
00:18:17:409104: error-drop
ip4-icmp-input: unknown type

Packet 5

00:18:18:409082: 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 0x5b60e3bb nsec 0x8ecad24 vlan 0 vlan_tpid 0
00:18:18:409087: ethernet-input
IP4: e2:0f:1e:59:ec:f7 -> 02:fe:d9:75:d5:b4
00:18:18:409091: ip4-input
ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3e66
    fragment id 0x262d

```

(continues on next page)

(continued from previous page)

```

ICMP echo_reply checksum 0x8e59
00:18:18:409093: 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 0x3e66
    fragment id 0x262d
ICMP echo_reply checksum 0x8e59
00:18:18:409096: ip4-local
    ICMP: 10.10.1.1 -> 10.10.1.2
        tos 0x00, ttl 64, length 84, checksum 0x3e66
        fragment id 0x262d
    ICMP echo_reply checksum 0x8e59
00:18:18:409098: ip4-icmp-input
ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3e66
    fragment id 0x262d
ICMP echo_reply checksum 0x8e59
00:18:18:409099: ip4-icmp-echo-reply
ICMP echo id 7531 seq 3
00:18:18:409102: ip4-drop
    ICMP: 10.10.1.1 -> 10.10.1.2
        tos 0x00, ttl 64, length 84, checksum 0x3e66
        fragment id 0x262d
    ICMP echo_reply checksum 0x8e59
00:18:18:409102: error-drop
ip4-icmp-input: unknown type

Packet 6

00:18:19:414750: 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 0x5b60e3bc nsec 0x92450f2 vlan 0 vlan_tpid 0
00:18:19:414754: ethernet-input
IP4: e2:0f:1e:59:ec:f7 -> 02:fe:d9:75:d5:b4
00:18:19:414757: ip4-input
ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3e52
    fragment id 0x2641
ICMP echo_reply checksum 0x9888
00:18:19:414760: 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 0x3e52
    fragment id 0x2641
ICMP echo_reply checksum 0x9888
00:18:19:414762: ip4-local
    ICMP: 10.10.1.1 -> 10.10.1.2
        tos 0x00, ttl 64, length 84, checksum 0x3e52
        fragment id 0x2641
    ICMP echo_reply checksum 0x9888
00:18:19:414764: ip4-icmp-input
ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3e52
    fragment id 0x2641
ICMP echo_reply checksum 0x9888

```

(continues on next page)

(continued from previous page)

```
00:18:19:414765: ip4-icmp-echo-reply
ICMP echo id 7531 seq 4
00:18:19:414768: ip4-drop
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3e52
    fragment id 0x2641
  ICMP echo_reply checksum 0x9888
00:18:19:414769: error-drop
ip4-icmp-input: unknown type

Packet 7

00:18:20:418038: 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 0x5b60e3bd nsec 0x937bcc2 vlan 0 vlan_tpid 0
00:18:20:418042: ethernet-input
IP4: e2:0f:1e:59:ec:f7 -> 02:fe:d9:75:d5:b4
00:18:20:418045: ip4-input
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3e47
    fragment id 0x264c
  ICMP echo_reply checksum 0xc0e8
00:18:20:418048: 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 0x3e47
  fragment id 0x264c
  ICMP echo_reply checksum 0xc0e8
00:18:20:418049: ip4-local
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3e47
    fragment id 0x264c
  ICMP echo_reply checksum 0xc0e8
00:18:20:418054: ip4-icmp-input
ICMP: 10.10.1.1 -> 10.10.1.2
  tos 0x00, ttl 64, length 84, checksum 0x3e47
  fragment id 0x264c
  ICMP echo_reply checksum 0xc0e8
00:18:20:418054: ip4-icmp-echo-reply
ICMP echo id 7531 seq 5
00:18:20:418057: ip4-drop
  ICMP: 10.10.1.1 -> 10.10.1.2
    tos 0x00, ttl 64, length 84, checksum 0x3e47
    fragment id 0x264c
  ICMP echo_reply checksum 0xc0e8
00:18:20:418058: error-drop
ip4-icmp-input: unknown type

Packet 8

00:18:21:419208: 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 0x5b60e3be nsec 0x92a9429 vlan 0 vlan_tpid 0
```

(continues on next page)

(continued from previous page)

```

00:18:21:419876: ethernet-input
ARP: e2:0f:1e:59:ec:f7 -> 02:fe:d9:75:d5:b4
00:18:21:419881: arp-input
request, type ethernet/IP4, address size 6/4
e2:0f:1e:59:ec:f7/10.10.1.1 -> 00:00:00:00:00:00/10.10.1.2
00:18:21:419896: host-vpplout-output
host-vpplout
ARP: 02:fe:d9:75:d5:b4 -> e2:0f:1e:59:ec:f7
reply, type ethernet/IP4, address size 6/4
02:fe:d9:75:d5:b4/10.10.1.2 -> e2:0f:1e:59:ec:f7/10.10.1.1

```

After examining the trace, clear it again using `vpp# clear trace`.

Examine arp tables

```

vpp# show ip arp

```

Time	IP4	Flags	Ethernet	Interface
1101.5636	10.10.1.1	D	e2:0f:1e:59:ec:f7	host-vpplout

Examine routing tables

```

vpp# show ip fib
  ipv4-VRF:0, fib_index:0, flow hash:[src dst sport dport proto ] locks:[src:plugin-
→hi:2, src:adjacency:1, src:default-route:1, ]
0.0.0.0/0
unicast-ip4-chain
[@0]: dpo-load-balance: [proto:ip4 index:1 buckets:1 uRPF:0 to:[0:0]]
  [0] [@0]: dpo-drop ip4
0.0.0.0/32
unicast-ip4-chain
[@0]: dpo-load-balance: [proto:ip4 index:2 buckets:1 uRPF:1 to:[0:0]]
  [0] [@0]: dpo-drop ip4
10.10.1.0/32
unicast-ip4-chain
[@0]: dpo-load-balance: [proto:ip4 index:10 buckets:1 uRPF:9 to:[0:0]]
  [0] [@0]: dpo-drop ip4
10.10.1.1/32
unicast-ip4-chain
[@0]: dpo-load-balance: [proto:ip4 index:13 buckets:1 uRPF:12 to:[5:420] via:[2:168]]
  [0] [@5]: ipv4 via 10.10.1.1 host-vpplout: mtu:9000 e20f1e59ecf702fed975d5b40800
10.10.1.0/24
unicast-ip4-chain
[@0]: dpo-load-balance: [proto:ip4 index:9 buckets:1 uRPF:8 to:[0:0]]
  [0] [@4]: ipv4-glean: host-vpplout: mtu:9000 ffffffff02fed975d5b40806
10.10.1.2/32
unicast-ip4-chain
[@0]: dpo-load-balance: [proto:ip4 index:12 buckets:1 uRPF:13 to:[7:588]]
  [0] [@2]: dpo-receive: 10.10.1.2 on host-vpplout
10.10.1.255/32
unicast-ip4-chain
[@0]: dpo-load-balance: [proto:ip4 index:11 buckets:1 uRPF:11 to:[0:0]]
  [0] [@0]: dpo-drop ip4
224.0.0.0/4

```

(continues on next page)

(continued from previous page)

```
unicast-ip4-chain
[@0]: dpo-load-balance: [proto:ip4 index:4 buckets:1 uRPF:3 to:[0:0]]
  [0] [@0]: dpo-drop ip4
240.0.0.0/4
unicast-ip4-chain
[@0]: dpo-load-balance: [proto:ip4 index:3 buckets:1 uRPF:2 to:[0:0]]
  [0] [@0]: dpo-drop ip4
255.255.255.255/32
unicast-ip4-chain
[@0]: dpo-load-balance: [proto:ip4 index:5 buckets:1 uRPF:4 to:[0:0]]
  [0] [@0]: dpo-drop ip4
```

5.1.7 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 FD.io VPP routing table

And revisit the old ones:

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

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 FD.io VPP routing table

And revisit the old ones:

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

FD.io VPP command learned in this exercise

1. `ip route add`

Topology

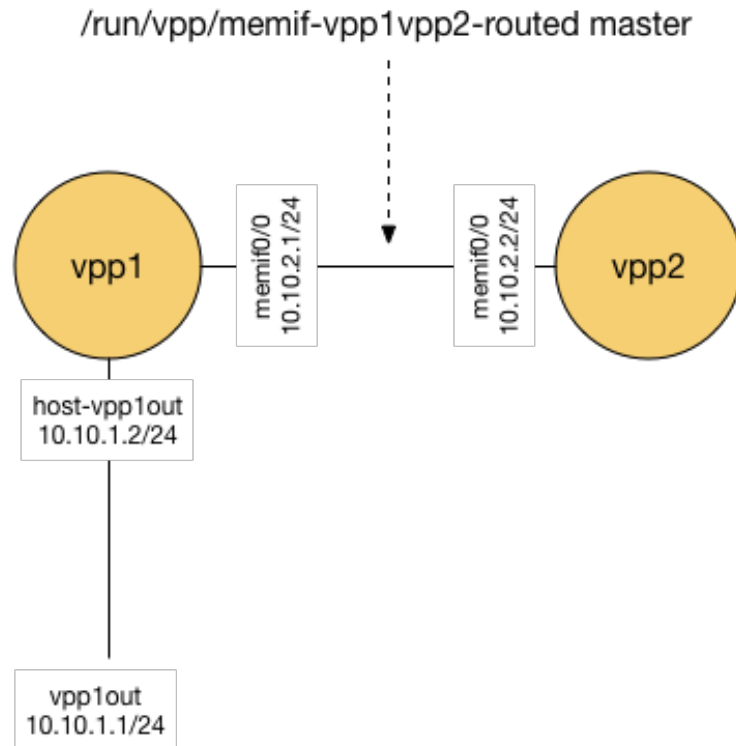


Fig. 2: Connect two FD.io VPP topology

Initial State

The initial state here is presumed to be the final state from the exercise [Connecting two FD.io VPP instances](#)

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
```

(continues on next page)

(continued from previous page)

```
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

```
vpp# 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

5.1.8 Connecting Two FD.io VPP Instances

memif is a very high performance, direct memory interface type which can be used between FD.io 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 FD.io VPP instances

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

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

Connecting two FD.io VPP Instances

memif is a very high performance, direct memory interface type which can be used between FD.io 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 FD.io VPP instances

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

1. Run a second FD.io VPP instance

2. Add an ip address to a FD.io VPP interface
3. Ping from FD.io VPP

Topology

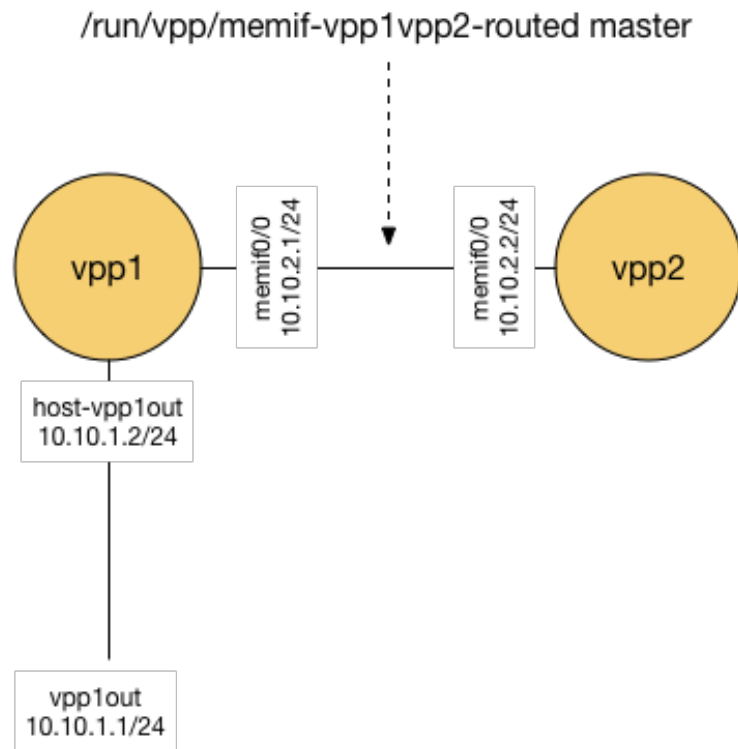


Fig. 3: Connect two FD.io VPP topology

Initial state

The initial state here is presumed to be the final state from the exercise [Create an Interface](#)

Running a second FD.io VPP instances

You should already have a FD.io VPP instance running named: vpp1.

Run a second FD.io VPP instance named: vpp2.

Create memif interface on vpp1

Create a memif interface on vpp1:

```
vpp# create interface 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 interface 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

Create memif interface on vpp2

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

```
vpp# create interface 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 interface 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

Ping from vpp1 to vpp2

Ping 10.10.2.2 from vpp1

```
$ ping 10.10.2.2
```

Ping 10.10.2.1 from vpp2

```
$ ping 10.10.2.1
```

5.1.9 Switching

Skills to be Learned

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

Switching

Skills to be Learned

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

FD.io VPP command learned in this exercise

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

Topology

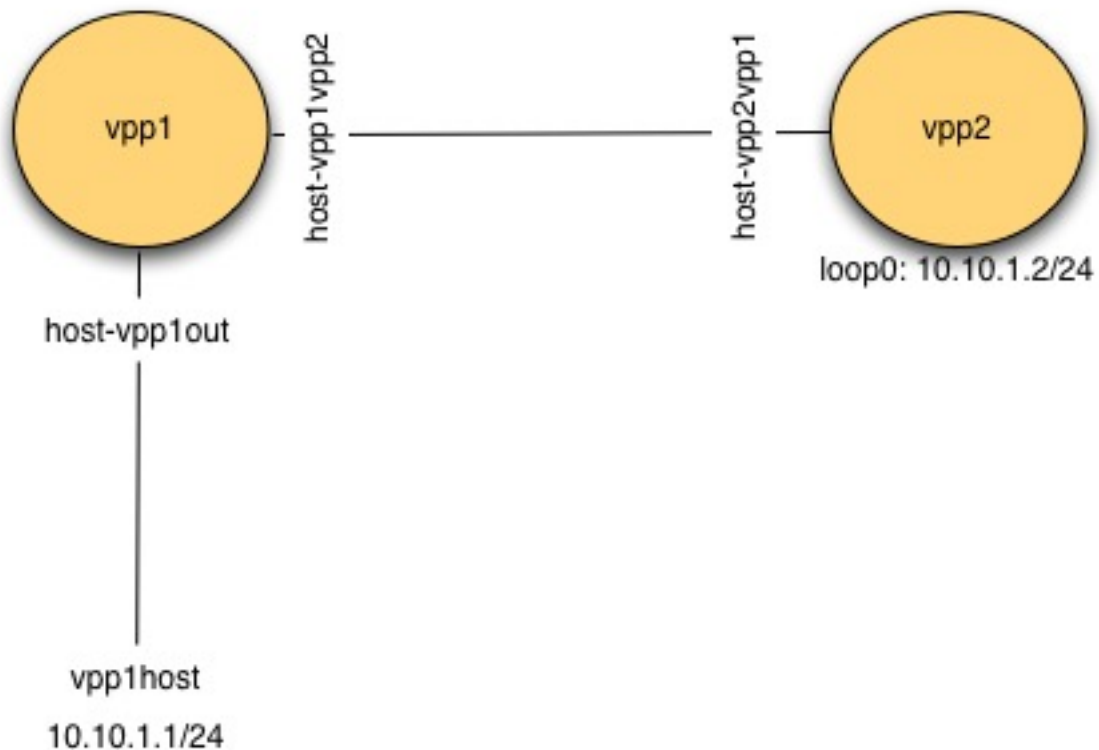


Fig. 4: Switching Topology

Initial state

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

Note: You will lose all your existing config in your FD.io 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
```

Run FD.io VPP instances

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

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

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.

Configure Bridge Domain on vpp1

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

```
vpp# 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.

```
vpp# set int l2 bridge host-vpp1out 0
vpp# set int l2 bridge host-vpp1vpp2 0
vpp# 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

```
vpp# set int l2 bridge host-vpp1out 1
```

Add host-vpp1vpp2 to bridge domain ID1

```
vpp# set int l2 bridge host-vpp1vpp2 1
```

Examine bridge domain 1:

```
vpp# 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

Configure loopback interface on vpp2

```
vpp# 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'

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

```
vpp# set int l2 bridge loop0 1 bvi
```

Add interface vpp2vpp1 to bridge domain 1

```
vpp# set int l2 bridge host-vpp2vpp1 1
```

Examine the bridge domain and interfaces.

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

Examine l2 fib

```
vpp# 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
```

```
vpp# 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
```

5.1.10 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

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

FD.io 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.

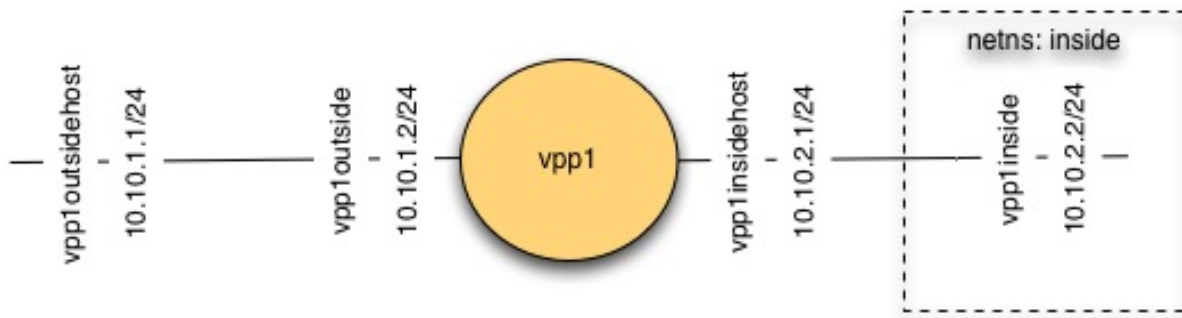


Fig. 5: SNAT Topology

Note: You will lose all your existing config in your FD.io 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
```

Install vpp-plugins

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

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

Create FD.io VPP instance

Create one FD.io VPP instance named vpp1.

Confirm snat plugin is present:

```
vpp# 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
```

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
```

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

Configure snat

Configure snat to use the address of host-vpp1outside

```
vpp# snat add interface address host-vpp1outside
```

Configure snat inside and outside interfaces

```
vpp# set interface snat in host-vpp1inside out host-vpp1outside
```

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

Ping via snat

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

Confirm snat

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

5.2 API User Guides

This chapter describes how to use the C, Python and java APIs.

5.2.1 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 management, please look at:

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

6.1 Conferences

6.1.1 August 2017 - Linuxcon NA 2016

Intro to fd.io

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

ONE-LISP Intro

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

6.1.2 June 2017 - Cisco Live US

VPP and container networking at 110Gbps

Event

This presentation was held during the on June 26th, 2017.

Speakers

Keith Burns

Slideshow

Presentation Powerpoint

Video

6.1.3 May 2017 - OSCON

40Gbps IPsec on commodity hardware

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

6.2 Summits

6.2.1 December 2017 - FD.io Mini Summit at KubeCon 2017

Arm Rallying the Ecosystem Around FD.io

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

Benchmarking and Analysis of Software Network Data Planes

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

Empowering the User Space Stack on Cloud Native Applications

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

VPP Host Stack

Event

This presentation was held during the FD.io Mini Summit at KubeCon 2017 on December 5th, 2017.

Speakers

Slideshow

Video

6.2.2 November 2017 - DPDK Summit

DPDK Summit is a DPDK community event held around the world.

VPP Host Stack

Event

Although packet forwarding with VPP and DPDK can now scale to tens of millions of packets per second per core, lack of alternatives to kernel-based sockets means that containers and host applications cannot take full advantage of this speed. To fill this gap, VPP was recently added functionality specifically designed to allow containerized or host applications to communicate via shared-memory if co-located, or via a high-performance TCP stack inter-host.

This presentation was held during the 2017 DPDK Summit on September 26th, 2017.

Speakers

- Florin Coras
- Dave Barach
- Keith Burns
- Dave Wallace

Slideshow

[Presentation PDF](#)

Video

[Video Presentation](#)

DPDK, VPP and pfSense 3.0

Event

pfSense is a open source firewall/vpn appliance, based on FreeBSD, started in 2006 with over 1M active installs. We are basing pfSense release 3.0 on FD.io's VPP, leveraging key DPDK components including cryptodev, while adding a CLI and RESTCONF layer, leveraging FRRouting and Strongswan.

This presentation was held during the 2017 DPDK Summit on September 26th, 2017.

Speakers

- Jim Thompson

Slideshow

[Presentation PDF](#)

Video

[Video Presentation](#)

6.2.3 September 2017 - Open Source Summit 2017

September 2017 - FD.io: The Universal Network Dataplane

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

6.2.4 FD.io Mini Summit

May 2017 - FD.io Mini Summit @ Open Stack, Boston

Performance Considerations for Packet Processing on Intel Architecture

Event

FD.io VPP is a modular, fast, scalable and deterministic data plane for NFV network workloads. Some common uses cases for VPP are as a vSwitch, a L3 Router, a vTEP, a Load Balancer, sockets applications and so much more, out of the box - no hacking required! It is optimized to take advantage of Intel Architecture Core and Platform accelerations to accelerate dataplane packet processing. It is permissively licensed as Apache 2, with a welcoming community of active developers behind it. It is extensible and can be used as an excellent set of building blocks to accelerate your next development project.

This presentation covers the following content: * Introduction to the architecture and design of VPP. * A walk through common VPP use cases, cloud and NFV integration. * Creating Network Nodes: accelerating VPP IPSEC with Intel Technologies. * Performance analysis on Intel Architecture.

This presentation was held during the on May 10th, 2017.

Speakers

- Patrick Lu
- Sergio Gonzalez Monroy

Slideshow

[Presentation Powerpoint](#)

Video

[Video Presentation](#)

6.2.5 April 2017 - UKNOF37

FD.io: building bespoke software data plane network functions

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

6.2.6 OPNFV

Listed are all the OPNFV Presentations related to FD.io.

February 2017 - OPNFV CSIT readout call

February 2017 - OPNFV CSIT Readout Call

FD.io CSIT project readout to OPNFV project

Slides

Used At The Event: Updated

June 2016 - OPNFV Summit

June 2016 - OPNFV Summit

Roadmap/Feedback Discussion

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

Session 1 - Introduction (DPDK, FD.io)

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

Session 2 - NFV Virtual Switching

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

Session 3 - SDN Controller/Orchestration

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

Session 4 - Data Plane Acceleration

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

Session 5 - Service Assurance

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

March 2016 - OPNFV Technical Workstream Meeting

March 2016 - OpNFV Technical Workstream

Introduction to FD.io at OPNFV

Event

This presentation was held during the OpNFV Technical Workstream Meeting on March 30th, 2016.

Speakers

Ed Warnicke

Slideshow

Presentation Powerpoint

Video

Ed Warnicke's Video Presentation

6.2.7 DPDK Summit

2017 DPDK Summit

2016 DPDK Summit

August 2016 - DPDK Summit

TLDK intro

Event

This presentation describes the Transport Layer Development Kit (TLDK), which is an FD.io project that provides a set of libraries to accelerate L4-L7 protocol processing for DPDK-based applications. The initial implementation supports UDP, with work on TCP in progress. The project scope also includes integration of TLDK libraries into Vector Packet Processing (VPP) graph nodes to provide a host stack. This presentation was held during the DPDK US Summit on August 10th, 2016.

Speakers

- Konstantin Ananyev

Slideshow

[Presentation Powerpoint](#)

Video

[Video Presentation](#)

6.2.8 March 2016 - Open Network Summit

FD.io is the Future

Event

This presentation was held during the Open Network Summit on March 15th, 2016.

Speakers

Ed Warnicke

Slideshow

[Presentation Powerpoint](#)

Video

[Ed Warnicke's Video Presentation](#)

Introduction to FD.io

Event

This presentation was held during the Open Network Summit on March 16th, 2016.

Speakers

Ed Warnicke

Slideshow

[Presentation Powerpoint](#)

Video

[Ed Warnicke's Video Presentation](#)

Packet Processed Storage in a Software Defined World

Event

This presentation was held during the Open Network Summit on March 17th, 2016.

Speakers

Ash Young

Slideshow

[Presentation Powerpoint](#)

Video

[Ash Young's Video Presentation](#)

6.2.9 February 2016 - OpenDaylight Design Forum

FD.io Intro

Event

This presentation was held during the OpenDaylight Design Forum on February 29th, 2016.

Speakers

Ed Warnicke

Slideshow

[Presentation Powerpoint](#)

Video

[Ed Warnicke's Video Presentation](#)

FD.io and Honeycomb

Event

This presentation was held during the OpenDaylight Design Forum on March 1st, 2016.

Speakers

Ed Warnicke

Slideshow

[Presentation Powerpoint](#)

Video

[Ed Warnicke's Video Presentation](#)

6.3 Meetings

6.3.1 July 2017 - FD.io VPP Terabit

FD.io: A Universal Terabit Network Dataplane

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

6.3.2 June 2017 - Intel SDN/NFV Dev Lab

FD.io Overview

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

Clear Containers + VPP Overview

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

6.3.3 August 2016 - TWS Meeting

Intro to DPDK Cryptodev and VPP PoC

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

6.3.4 May 2016 - FD.io Webinar

FD.io is the Future of Software Dataplanes

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

6.3.5 April 2016 - TSC Meeting

2016 May 31st Paris FD.io event proposal

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

6.4 Calls

6.4.1 2016 November - TLDK Call

TCP timer implementation overview

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

6.4.2 April 2017 - VPP Call

Building In Containers PoC

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

6.5 Fd.io Training Event

The FD.io Training provides new users with a detailed set of training courses on FD.io separated by days.

6.5.1 April 2016 - FD.io Training

Faster

Event

This presentation was held during the FD.io Training/Hackfest on April 5th, 2016.

Speakers

Dave Ward

Slideshow

[Presentation Powerpoint](#)

Video

[Dave Ward's Video Presentation](#)

All Technical Content

As each session is delivered, the content will be provided here initially as PDF, with links to Videos, other content, to come as available. These presentations were held during the FD.io Training/Hackfest on April 5th, 2016.

Day One

Day One

List of FD.io presentations that were held in the FD.io Training/Hackfest on April 5th, 2016.

FD.io DevBoot: 1.2 VPP overview

Event

This presentation was held during the FD.io Training/Hackfest on April 5th, 2016.

Speakers

Dave Barach and Keith Burns

Slideshow

[Presentation Powerpoint](#)

Video

DevBoot Presentation

FD.io DevBoot: 1.3 Container Demo

Event

This presentation was held during the FD.io Training/Hackfest on April 5th, 2016.

Speakers

Nikos Bregiannis

Slideshow

Presentation Powerpoint

Video

DevBoot Presentation

FD.io DevBoot: 1.4 Code Contribution Mechanics

Event

This presentation was held during the FD.io Training/Hackfest on April 5th, 2016.

Speakers

Keith Burns

Slideshow

Presentation Powerpoint

Video

DevBoot Presentation

FD.io DevBoot: 1.5 DPDK Introduction

Event

This presentation was held during the FD.io Training/Hackfest on April 5th, 2016.

Speakers

Cristian Dumitrescu

Slideshow

Presentation Powerpoint

Video

DevBoot Presentation

Day Two

Day Two

FD.io DevBoot: Day 2 Pre-work and scripts

Event

This presentation was held on April, 2016.

Preparing for Day 2

1. *VM's with Vagrant*
2. Build vagrant environment

```
$ vagrant ssh
$ sudo su
$ wget -O /vagrant/netns.sh "https://tinyurl.com/devboot-netns"
$ wget -O /vagrant/macswap.conf "https://tinyurl.com/devboot-macswap-conf"
$ wget -O ~/.gdbinit "https://tinyurl.com/devboot-gdbinit"
```

Note: For Init section: Breakpoints High-level chicken-scrawl of init

Mac Swap Breakdown

Day Three

Day Three

Pre Work

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

2 4 more macswap

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

3 1 buffer metadata

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

3 2 multi core

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

DPDK Optimizations

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

Supplemental

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

Day Four

Day Four

FD.io DevBoot: 4.1: Map in VPP

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

FD.io DevBoot: 4.2: Project: ONE LISP

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

FD.io DevBoot: 4.3 CSIT + Performance testing

Event

This presentation was held during the on th, 201.

Speakers

Slideshow

Video

7.1 VM's with Vagrant

This reference guide will cover using Vagrant to boot a VM (virtual machine).

7.1.1 Overview

This guide shows how the VPP *Vagrant File* interacts with shell scripts to configure a *Vagrant box* that boots a VM with VPP.

Prerequisites

You have the [FD.io VPP repo](#) cloned locally on your machine.

This guide will refer to the following files from that repo: *Vagrantfile*, *build.sh*, *env.sh*, and *update.sh*.

7.1.2 Installing Vbox and Vagrant

Installing VirtualBox

First download VirtualBox, which is virtualization software for creating VM's.

If you're on CentOS, follow the [steps here](#).

If you're on Ubuntu, perform:

```
$ sudo apt-get install virtualbox
```

Installing Vagrant

Here we are on a 64-bit version of CentOS, downloading and installing Vagrant 2.1.2:

```
$ yum -y install https://releases.hashicorp.com/vagrant/2.1.2/vagrant_2.1.2_x86_64.rpm
```

This is a similar command, but on a 64-bit version of Debian:

```
$ sudo apt-get install https://releases.hashicorp.com/vagrant/2.1.2/vagrant_2.1.2_x86_64.deb
```

If you want to download a newer version of Vagrant or one specific to your OS and architecture, go to the [Vagrant download page](#), right-click and copy the link address for your specified version, and replace the above install command for your respective OS and architecture.

7.1.3 Setting your ENV Variables

The *Vagrant File* used in the VPP repo sets the configuration options based on your ENV (environment) variables, or to default the configuration at specified values if your ENV variables are not initialized (if you did not run the *env.sh* script found below).

This is the *env.sh* script found in *vpp/extras/vagrant*. When run, the script sets ENV variables using the **export** command.

```
export VPP_VAGRANT_DISTRO="ubuntu1604"
export VPP_VAGRANT_NICS=2
export VPP_VAGRANT_VMCPUS=4
export VPP_VAGRANT_VMRAM=4096
```

In the *Vagrant File*, you can see these same ENV variables used (discussed on the next page).

Adding your own ENV variables is easy. For example, if you wanted to setup proxies for your VM, you would add to this file the **export** commands found in the *building VPP commands section*. Note that this only works if the ENV variable is defined in the *Vagrant File*.

Once you're finished with *env.sh* script, and you are in the directory containing *env.sh*, run the script to set the ENV variables with:

```
$ source ./env.sh
```

7.1.4 Vagrantfiles

A *Vagrantfile* contains the box and provision configuration settings for your VM. The syntax of Vagrantfiles is Ruby (Ruby experience is not necessary).

The command **vagrant up** creates a *Vagrant Box* based on your Vagrantfile. A Vagrant box is one of the motivations for using Vagrant - its a "development-ready box" that can be copied to other machines to recreate the same environment.

It's common for people to think that a Vagrant box *is* the VM. But rather, the VM is *inside* a Vagrant box, with the box containing additional configuration options you can set, such as VM options, scripts to run on boot, etc.

This [Vagrant website for boxes](#) shows you how to configure a basic Vagrantfile for your specific OS and VM software.

Box configuration

Looking at the *Vagrant File*, we can see that the default OS is Ubuntu 16.04 (since the variable *distro* equals *ubuntu1604* if there is no *VPP_VAGRANT_DISTRO* variable set - thus the **else** case is executed.)

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

Vagrant.configure(2) do |config|

  # Pick the right distro and bootstrap, default is ubuntu1604
  distro = ( ENV['VPP_VAGRANT_DISTRO'] || "ubuntu1604")
  if distro == 'centos7'
    config.vm.box = "centos/7"
    config.vm.box_version = "1708.01"
    config.ssh.insert_key = false
  elsif distro == 'opensuse'
    config.vm.box = "opensuse/openSUSE-42.3-x86_64"
    config.vm.box_version = "1.0.4.20170726"
  else
    config.vm.box = "puppetlabs/ubuntu-16.04-64-nocm"
```

As mentioned in the previous page, you can specify which OS and VM provider you want for your Vagrant box from the [Vagrant boxes page](#), and setting your ENV variable appropriately in *env.sh*.

Next in the Vagrantfile, you see some *config.vm.provision* commands. As paraphrased from [Basic usage of Provisioners](#), by default these are only run *once* - during the first boot of the box.

```
config.vm.provision :shell, :path => File.join(File.dirname(__FILE__), "update.sh")
config.vm.provision :shell, :path => File.join(File.dirname(__FILE__), "build.sh"), _
↳:args => "/vpp vagrant"
```

The two lines above set the VM to run two scripts during its first bootup: an update script *update.sh* that does basic updating and installation of some useful tools, as well as *build.sh* that builds (but does **not** install) VPP in the VM. You can view these scripts on your own for more detail on the commands used.

Looking further in the *Vagrant File*, you can see more Ruby variables being set to ENV's or to a default value:

```
# Define some physical ports for your VMs to be used by DPDK
nics = (ENV['VPP_VAGRANT_NICS'] || "2").to_i(10)
for i in 1..nics
  config.vm.network "private_network", type: "dhcp"
end

# use http proxy if available
if ENV['http_proxy'] && Vagrant.has_plugin?("vagrant-proxyconf")
  config.proxy.http      = ENV['http_proxy']
  config.proxy.https     = ENV['https_proxy']
  config.proxy.no_proxy  = "localhost,127.0.0.1"
end

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

You can see how the box or VM is configured, such as the amount of NICs (defaults to 3 NICs: 1 x NAT - host access and 2 x VPP DPDK enabled), CPUs (defaults to 2), and RAM (defaults to 4096 MB).

Box bootup

Once you're satisfied with your *Vagrantfile*, boot the box with:

```
$ vagrant up
```

Doing this above command will take quite some time, since you are installing a VM and building VPP. Take a break and get some scooby snacks while you wait.

To confirm it is up, show the status and information of Vagrant boxes with:

```
$ vagrant global-status
id            name      provider  state    directory
-----
d90a17b      default  virtualbox poweroff  /home/centos/andrew-vpp/vppsb/vpp-userdemo
77b085e      default  virtualbox poweroff  /home/centos/andrew-vpp/vppsb2/vpp-userdemo
c1c8952      default  virtualbox poweroff  /home/centos/andrew-vpp/testingVPPSB/extras/
c199140      default  virtualbox running   /home/centos/andrew-vpp/vppsb3/vpp-userdemo

You will have only one machine running, but I have multiple as shown above.
```

Note: To poweroff your VM, type:

```
$ vagrant halt <id>
```

To resume your VM, type:

```
$ vagrant resume <id>
```

To destroy your VM, type:

```
$ vagrant destroy <id>
```

Note that “destroying” a VM does not erase the box, but rather destroys all resources allocated for that VM. For other Vagrant commands, such as destroying a box, refer to the [Vagrant CLI Page](#).

7.1.5 Accessing your VM

ssh into the newly created box:

```
$ vagrant ssh <id>
```

Sample output looks like:

```
$ vagrant ssh c1c
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0-21-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Mon Jun 25 08:05:38 2018 from 10.0.2.2
vagrant@localhost:~$
```

Note: Type **exit** in the command-line if you want to exit the VM.

Become the root with:

```
$ sudo bash
```

Now *install* VPP in the VM. Keep in mind that VPP is already built (but not yet installed) at this point based on the commands from the provisioned script *build.sh*.

When you ssh into your Vagrant box you will be placed in the directory */home/vagrant*. Change directories to */vpp/build-root*, and run these commands to install VPP based on your OS and architecture:

For Ubuntu systems:

```
# dpkg -i *.deb
```

For CentOS systems:

```
# rpm -Uvh *.rpm
```

Since VPP is now installed, you can start running VPP with:

```
# service vpp start
```

7.1.6 Vagrant File

This is the Vagrantfile provided in the Git VPP repo.

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

Vagrant.configure(2) do |config|

  # Pick the right distro and bootstrap, default is ubuntu1604
  distro = ( ENV['VPP_VAGRANT_DISTRO'] || "ubuntu1604" )
  if distro == 'centos7'
    config.vm.box = "centos/7"
    config.vm.box_version = "1708.01"
    config.ssh.insert_key = false
  elsif distro == 'opensuse'
    config.vm.box = "opensuse/openSUSE-42.3-x86_64"
    config.vm.box_version = "1.0.4.20170726"
  else
    config.vm.box = "puppetlabs/ubuntu-16.04-64-nocm"
  end
  config.vm.box_check_update = false

  config.vm.provision :shell, :path => File.join(File.dirname(__FILE__), "update.sh")
  config.vm.provision :shell, :path => File.join(File.dirname(__FILE__), "build.sh"),
  ↪:args => "/vpp vagrant"

  post_build = ( ENV['VPP_VAGRANT_POST_BUILD'] )
  if post_build == "test"
    config.vm.provision "shell", inline: "echo Testing VPP; cd /vpp; make test"
  elsif post_build == "install"
```

(continues on next page)

(continued from previous page)

```

    config.vm.provision :shell, :path => File.join(File.dirname(__FILE__), "install.sh"
↳"), :args => "/vpp"
    config.vm.provision :shell, :path => File.join(File.dirname(__FILE__),
↳"clearinterfaces.sh")
    config.vm.provision :shell, :path => File.join(File.dirname(__FILE__), "run.sh")
end

# Add .gnupg dir in so folks can sign patches
# Note, as gnupg puts socket files in that dir, we have
# to be cautious and make sure we are dealing with a plain file
homedir = File.expand_path("~/")
Dir["#{homedir}/.gnupg/**/*"].each do |fname|
  if File.file?(fname)
    destname = fname.sub(Regexp.escape("#{homedir}/"), '')
    config.vm.provision "file", source: fname, destination: destname
  end
end

# Copy in the .gitconfig if it exists
if File.file?(File.expand_path("~/gitconfig"))
  config.vm.provision "file", source: "~/gitconfig", destination: ".gitconfig"
end

# vagrant-cachier caches apt/yum etc to speed subsequent
# vagrant up
# to enable, run
# vagrant plugin install vagrant-cachier
#
if Vagrant.has_plugin?("vagrant-cachier")
  config.cache.scope = :box
end

# Define some physical ports for your VMs to be used by DPDK
nics = (ENV['VPP_VAGRANT_NICS'] || "2").to_i(10)
for i in 1..nics
  config.vm.network "private_network", type: "dhcp"
end

# use http proxy if available
if ENV['http_proxy'] && Vagrant.has_plugin?("vagrant-proxyconf")
  config.proxy.http      = ENV['http_proxy']
  config.proxy.https     = ENV['https_proxy']
  config.proxy.no_proxy  = "localhost,127.0.0.1"
end

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

config.ssh.forward_agent = true
config.ssh.forward_x11 = true

config.vm.provider "virtualbox" do |vb|
  vb.customize ["modifyvm", :id, "--ioapic", "on"]
  vb.memory = "#{vmram}"
  vb.cpus = "#{vmcpu}"

  # rsync the vpp directory if provision hasn't happened yet

```

(continues on next page)

(continued from previous page)

```

unless File.exist? (".vagrant/machines/default/virtualbox/action_provision")
  config.vm.synced_folder "../..", "/vpp", type: "rsync",
    rsync__auto: false,
    rsync__exclude: [
      "build-root/build*/",
      "build-root/install*/",
      "build-root/images*/",
      "build-root/*.deb",
      "build-root/*.rpm",
      "build-root/*.changes",
      "build-root/python",
      "build-root/deb/debian/*.dkms",
      "build-root/deb/debian/*.install",
      "build-root/deb/debian/changes",
      "build-root/tools"]
end

#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
config.vm.provider "vmware_fusion" do |fusion,override|
  fusion.vmx["memsize"] = "#{vmram}"
  fusion.vmx["numvcpus"] = "#{vmcpu}"
end
config.vm.provider "libvirt" do |lv|
  lv.memory = "#{vmram}"
  lv.cpus = "#{vmcpu}"
end
config.vm.provider "vmware_workstation" do |vws,override|
  vws.vmx["memsize"] = "#{vmram}"
  vws.vmx["numvcpus"] = "#{vmcpu}"
end
end
end

```

7.2 Command Line Reference

This is a reference guide for the vpp debug commands that are referenced in the within these documents. This is **NOT** a complete list. For a complete list refer to the Debug CLI section of the [Source Code Documents](#).

The debug CLI can be executed from a su shell using the vppctl command.

```

# sudo bash
# vppctl show interface

```

Name	Idx	State	Counter	Count
TenGigabitEthernet86/0/0	1	up	rx packets	6569213
			rx bytes	9928352943
			tx packets	50384
			tx bytes	3329279
TenGigabitEthernet86/0/1	2	down		
VirtualEthernet0/0/0	3	up	rx packets	50384
			rx bytes	3329279
			tx packets	6569213
			tx bytes	9928352943

(continues on next page)

(continued from previous page)

local0	0	down	drops	1498
--------	---	------	-------	------

Commands can also be executed from the vppct shell.

```
# vppctl

  _/ _/ _ \  ( ) _  | | / / _ \ / _ \
 _/ _// // // / _ \ | | / / _ / _ /
 /_ / _ _ ( ) _ \ _ / | _ / _ / _ /

vpp# show interface

      Name                               Idx      State      Counter      Count
TenGigabitEthernet86/0/0                1         up      rx packets      6569213
                                         rx bytes      9928352943
                                         tx packets       50384
                                         tx bytes      3329279
TenGigabitEthernet86/0/1                2         down
VirtualEthernet0/0/0                    3         up      rx packets       50384
                                         rx bytes      3329279
                                         tx packets      6569213
                                         tx bytes      9928352943
                                         drops           1498
local0                                   0         down
```

7.2.1 Interface Commands

Show Hardware-Interfaces

Display more detailed information about all or a list of given interfaces. The verbosity of the output can be controlled by the following optional parameters:

- brief: Only show name, index and state (default for bonded interfaces).
- verbose: Also display additional attributes (default for all other interfaces).
- detail: Also display all remaining attributes and extended statistics.

To limit the output of the command to bonded interfaces and their slave interfaces, use the “*bond*” optional parameter.

Summary/Usage

```
show hardware-interfaces [brief|verbose|detail] [bond] [<interface> [<interface> [...]]]
[<sw_idx> [<sw_idx> [...]]].
```

Examples

Example of how to display default data for all interfaces:


```
vpp# show hardware-interfaces
      Name                               Idx  Link  Hardware
GigabitEthernet7/0/0                    1    up   GigabitEthernet7/0/0
  Ethernet address ec:f4:bb:c0:bc:fc
  Intel e1000
    carrier up full duplex speed 1000 mtu 9216
    rx queues 1, rx desc 1024, tx queues 3, tx desc 1024
    cpu socket 0
GigabitEthernet7/0/1                    2    up   GigabitEthernet7/0/1
  Ethernet address ec:f4:bb:c0:bc:fd
  Intel e1000
    carrier up full duplex speed 1000 mtu 9216
    rx queues 1, rx desc 1024, tx queues 3, tx desc 1024
    cpu socket 0
VirtualEthernet0/0/0                    3    up   VirtualEthernet0/0/0
  Ethernet address 02:fe:a5:a9:8b:8e
VirtualEthernet0/0/1                    4    up   VirtualEthernet0/0/1
  Ethernet address 02:fe:c0:4e:3b:b0
VirtualEthernet0/0/2                    5    up   VirtualEthernet0/0/2
  Ethernet address 02:fe:1f:73:92:81
VirtualEthernet0/0/3                    6    up   VirtualEthernet0/0/3
  Ethernet address 02:fe:f2:25:c4:68
local0                                  0    down local0
  local
```

Example of how to display ‘*verbose*’ data for an interface by name and software index (where 2 is the software index):

```
vpp# show hardware-interfaces GigabitEthernet7/0/0 2 verbose
      Name                               Idx  Link  Hardware
GigabitEthernet7/0/0                    1    up   GigabitEthernet7/0/0
  Ethernet address ec:f4:bb:c0:bc:fc
  Intel e1000
    carrier up full duplex speed 1000 mtu 9216
    rx queues 1, rx desc 1024, tx queues 3, tx desc 1024
    cpu socket 0
GigabitEthernet7/0/1                    2    down GigabitEthernet7/0/1
  Ethernet address ec:f4:bb:c0:bc:fd
  Intel e1000
    carrier up full duplex speed 1000 mtu 9216
    rx queues 1, rx desc 1024, tx queues 3, tx desc 1024
    cpu socket 0
```

Clear Hardware-Interfaces

Clear the extended statistics for all or a list of given interfaces (statistics associated with the ‘*show hardware-interfaces*’ command).

Summary/Usage

```
clear hardware-interfaces [<interface> [<interface> [...]] [<sw_idx> [<sw_idx> [...]]].
```

Examples

Example of how to clear the extended statistics for all interfaces:

```
vpp# clear hardware-interfaces
```

Example of how to clear the extended statistics for an interface by name and software index (where 2 is the software index):

```
vpp# clear hardware-interfaces GigabitEthernet7/0/0 2
```

Interface Commands

Show Interface

Shows software interface information including counters and features

Summary/Usage

```
show interface [address|addr|features|feat] [<interface> [<interface> [...]]]
```

Examples

Example of how to show the interface counters:

```
vpp# show int
```

Name	Idx	State	Counter	Count
TenGigabitEthernet86/0/0	1	up	rx packets	6569213
			rx bytes	9928352943
			tx packets	50384
			tx bytes	3329279
TenGigabitEthernet86/0/1	2	down		
VirtualEthernet0/0/0	3	up	rx packets	50384
			rx bytes	3329279
			tx packets	6569213
			tx bytes	9928352943
			drops	1498
local0	0	down		

Example of how to display the interface placement:

```
vpp# show interface rx-placement
Thread 1 (vpp_wk_0):
  node dpdk-input:
    GigabitEthernet7/0/0 queue 0 (polling)
  node vhost-user-input:
    VirtualEthernet0/0/12 queue 0 (polling)
    VirtualEthernet0/0/12 queue 2 (polling)
    VirtualEthernet0/0/13 queue 0 (polling)
    VirtualEthernet0/0/13 queue 2 (polling)
Thread 2 (vpp_wk_1):
  node dpdk-input:
```

(continues on next page)

(continued from previous page)

```
GigabitEthernet7/0/1 queue 0 (polling)
node vhost-user-input:
  VirtualEthernet0/0/12 queue 1 (polling)
  VirtualEthernet0/0/12 queue 3 (polling)
  VirtualEthernet0/0/13 queue 1 (polling)
  VirtualEthernet0/0/13 queue 3 (polling)
```

Clear Interfaces

Clear the statistics for all interfaces (statistics associated with the *'show interface'* command).

Summary/Usage

```
clear interfaces
```

Example

Example of how to clear the statistics for all interfaces:

```
vpp# clear interfaces
```

Set Interface Mac Address

The *'set interface mac address'* command allows to set MAC address of given interface. In case of NIC interfaces the one has to support MAC address change. A side effect of MAC address change are changes of MAC addresses in FIB tables (ipv4 and ipv6).

Summary/Usage

```
set interface mac address <interface> <mac-address>.
```

Examples

Examples of how to change MAC Address of interface:

```
vpp# set interface mac address GigabitEthernet0/8/0 aa:bb:cc:dd:ee:01
vpp# set interface mac address host-vpp0 aa:bb:cc:dd:ee:02
vpp# set interface mac address tap-0 aa:bb:cc:dd:ee:03
vpp# set interface mac address pg0 aa:bb:cc:dd:ee:04
```

Set Interface Mtu

Summary/Usage

```
set interface mtu [packet|ip4|ip6|mpls] <value> <interface>.
```

Set Interface Promiscuous

Summary/Usage

```
set interface promiscuous [on|off] <interface>.
```

Set Interface State

This command is used to change the admin state (up/down) of an interface.

If an interface is down, the optional *'punt'* flag can also be set. The *'punt'* flag implies the interface is disabled for forwarding but punt all traffic to slow-path. Use the *'enable'* flag to clear *'punt'* flag (interface is still down).

Summary/Usage

```
set interface state <interface> [up|down|punt|enable].
```

Examples

Example of how to configure the admin state of an interface to **up**:

```
vpp# set interface state GigabitEthernet2/0/0 up
```

Example of how to configure the admin state of an interface to **down**:

```
vpp# set interface state GigabitEthernet2/0/0 down
```

Create Sub-Interfaces

This command is used to add VLAN IDs to interfaces, also known as subinterfaces. The primary input to this command is the *'interface'* and *'subId'* (subinterface Id) parameters. If no additional VLAN ID is provide, the VLAN ID is assumed to be the *'subId'*. The VLAN ID and *'subId'* can be different, but this is not recommended.

This command has several variations:

- **create sub-interfaces <interface> <subId>** - Create a subinterface to process packets with a given 802.1q VLAN ID (same value as the *'subId'*).
- **create sub-interfaces <interface> <subId> default** - Adding the *'default'* parameter indicates that packets with VLAN IDs that do not match any other subinterfaces should be sent to this subinterface.
- **create sub-interfaces <interface> <subId> untagged** - Adding the *'untagged'* parameter indicates that packets no VLAN IDs should be sent to this subinterface.

- **create sub-interfaces** <interface> <subId>-<subId> - Create a range of subinterfaces to handle a range of VLAN IDs.
- **create sub-interfaces** <interface> <subId> dot1q|dot1ad <vlanId>|any [exact-match] - Use this command to specify the outer VLAN ID, to either be explicit or to make the VLAN ID different from the 'subId'.
- **create sub-interfaces** <interface> <subId> dot1q|dot1ad <vlanId>|any inner-dot1q <vlanId>|any [exact-match] - Use this command to specify the outer VLAN ID and the inner VLAN ID.

When 'dot1q' or 'dot1ad' is explicitly entered, subinterfaces can be configured as either exact-match or non-exact match. Non-exact match is the CLI default. If 'exact-match' is specified, packets must have the same number of VLAN tags as the configuration. For non-exact-match, packets must at least that number of tags. L3 (routed) interfaces must be configured as exact-match. L2 interfaces are typically configured as non-exact-match. If 'dot1q' or 'dot1ad' is NOT entered, then the default behavior is exact-match.

Use the 'show interface' command to display all subinterfaces.

Summary/Usage

```
create sub-interfaces <interface> {<subId> [default|untagged]} | {<subId>-<subId>} | {
  <subId> dot1q|dot1ad <vlanId>|any [inner-dot1q <vlanId>|any] [exact-match]}.
```

Examples

Example of how to create a VLAN subinterface 11 to process packets on 802.1q VLAN ID 11:

```
vpp# create sub-interfaces GigabitEthernet2/0/0 11
```

The previous example is shorthand and is equivalent to:

```
vpp# create sub-interfaces GigabitEthernet2/0/0 11 dot1q 11 exact-match
```

Example of how to create a subinterface number that is different from the VLAN ID:

```
vpp# create sub-interfaces GigabitEthernet2/0/0 11 dot1q 100
```

Examples of how to create q-in-q and q-in-any subinterfaces:

```
vpp# create sub-interfaces GigabitEthernet2/0/0 11 dot1q 100 inner-dot1q 200
vpp# create sub-interfaces GigabitEthernet2/0/0 12 dot1q 100 inner-dot1q any
```

Examples of how to create dot1ad interfaces:

```
vpp# create sub-interfaces GigabitEthernet2/0/0 11 dot1ad 11
vpp# create sub-interfaces GigabitEthernet2/0/0 12 dot1ad 100 inner-dot1q 200
```

Examples of 'exact-match' versus non-exact match. A packet with outer VLAN 100 and inner VLAN 200 would match this interface, because the default is non-exact match:

```
vpp# create sub-interfaces GigabitEthernet2/0/0 5 dot1q 100
```

However, the same packet would NOT match this interface because 'exact-match' is specified and only one VLAN is configured, but packet contains two VLANs:

```
vpp# create sub-interfaces GigabitEthernet2/0/0 5 dot1q 100 exact-match
```

Example of how to create a subinterface to process untagged packets:

```
vpp# create sub-interfaces GigabitEthernet2/0/0 5 untagged
```

Example of how to create a subinterface to process any packet with a VLAN ID that does not match any other subinterface:

```
vpp# create sub-interfaces GigabitEthernet2/0/0 7 default
```

When subinterfaces are created, they are in the down state. Example of how to enable a newly created subinterface:

```
vpp# set interface GigabitEthernet2/0/0.7 up
```

7.2.2 Vhost User Commands

Create Vhost-User

Create a vHost User interface. Once created, a new virtual interface will exist with the name *'VirtualEthernet0/0/x'*, where 'x' is the next free index.

There are several parameters associated with a vHost interface:

- **socket <socket-filename>** - Name of the linux socket used by hypervisor and VPP to manage the vHost interface. If in *'server'* mode, VPP will create the socket if it does not already exist. If in *'client'* mode, hypervisor will create the socket if it does not already exist. The VPP code is indifferent to the file location. However, if SELinux is enabled, then the socket needs to be created in *'/var/run/vpp/'*.
- **server** - Optional flag to indicate that VPP should be the server for the linux socket. If not provided, VPP will be the client. In *'server'* mode, the VM can be reset without tearing down the vHost Interface. In *'client'* mode, VPP can be reset without bringing down the VM and tearing down the vHost Interface.
- **feature-mask <hex>** - Optional virtio/vhost feature set negotiated at startup. **This is intended for debugging only.** It is recommended that this parameter not be used except by experienced users. By default, all supported features will be advertised. Otherwise, provide the set of features desired.
 - 0x000008000 (15) - VIRTIO_NET_F_MRG_RXBUF
 - 0x000020000 (17) - VIRTIO_NET_F_CTRL_VQ
 - 0x000200000 (21) - VIRTIO_NET_F_GUEST_ANNOUNCE
 - 0x000400000 (22) - VIRTIO_NET_F_MQ
 - 0x004000000 (26) - VHOST_F_LOG_ALL
 - 0x008000000 (27) - VIRTIO_F_ANY_LAYOUT
 - 0x010000000 (28) - VIRTIO_F_INDIRECT_DESC
 - 0x040000000 (30) - VHOST_USER_F_PROTOCOL_FEATURES
 - 0x100000000 (32) - VIRTIO_F_VERSION_1
- **hwaddr <mac-addr>** - Optional ethernet address, can be in either X:X:X:X:X:X unix or X.X.X cisco format.
- **renumber <dev_instance>** - Optional parameter which allows the instance in the name to be specified. If instance already exists, name will be used anyway and multiple instances will have the same name. Use with caution.

Summary/Usage

```
create vhost-user socket <socket-filename> [server] [feature-mask <hex>] [hwaddr <mac-
→addr>] [renumber <dev_instance>]
```

Examples

Example of how to create a vhost interface with VPP as the client and all features enabled:

```
vpp# create vhost-user socket /var/run/vpp/vhost1.sock
VirtualEthernet0/0/0
```

Example of how to create a vhost interface with VPP as the server and with just multiple queues enabled:

```
vpp# create vhost-user socket /var/run/vpp/vhost2.sock server feature-mask 0x40400000
VirtualEthernet0/0/1
```

Once the vHost interface is created, enable the interface using:

```
vpp# set interface state VirtualEthernet0/0/0 up
```

Show Vhost-User

Display the attributes of a single vHost User interface (provide interface name), multiple vHost User interfaces (provide a list of interface names separated by spaces) or all Vhost User interfaces (omit an interface name to display all vHost interfaces).

Summary/Usage

```
show vhost-user [<interface> [<interface> [...]] [descriptors].
```

Examples

Example of how to display a vhost interface:

```
vpp# show vhost-user VirtualEthernet0/0/0
Virtio vhost-user interfaces
Global:
  coalesce frames 32 time 1e-3
Interface: VirtualEthernet0/0/0 (ifindex 1)
virtio_net_hdr_sz 12
features mask (0xffffffffffffffff):
features (0x50408000):
  VIRTIO_NET_F_MRG_RXBUF (15)
  VIRTIO_NET_F_MQ (22)
  VIRTIO_F_INDIRECT_DESC (28)
  VHOST_USER_F_PROTOCOL_FEATURES (30)
protocol features (0x3)
  VHOST_USER_PROTOCOL_F_MQ (0)
  VHOST_USER_PROTOCOL_F_LOG_SHMFD (1)
```

(continues on next page)

(continued from previous page)

```

socket filename /var/run/vpp/vhost1.sock type client errno "Success"

rx placement:
  thread 1 on vring 1
  thread 1 on vring 5
  thread 2 on vring 3
  thread 2 on vring 7
tx placement: spin-lock
  thread 0 on vring 0
  thread 1 on vring 2
  thread 2 on vring 0

Memory regions (total 2)
region fd      guest_phys_addr    memory_size      userspace_addr    mmap_offset      _
↳ mmap_addr
=====
↳ =====
  0      60      0x0000000000000000 0x00000000000a0000 0x00002aaaaac00000_
↳ 0x0000000000000000 0x00002aab2b400000
  1      61      0x00000000000c0000 0x000000003ff40000 0x00002aaaaacc0000_
↳ 0x00000000000c0000 0x00002aababcc0000

Virtqueue 0 (TX)
  qsz 256 last_avail_idx 0 last_used_idx 0
  avail.flags 1 avail.idx 128 used.flags 1 used.idx 0
  kickfd 62 callfd 64 errfd -1

Virtqueue 1 (RX)
  qsz 256 last_avail_idx 0 last_used_idx 0
  avail.flags 1 avail.idx 0 used.flags 1 used.idx 0
  kickfd 65 callfd 66 errfd -1

Virtqueue 2 (TX)
  qsz 256 last_avail_idx 0 last_used_idx 0
  avail.flags 1 avail.idx 128 used.flags 1 used.idx 0
  kickfd 63 callfd 70 errfd -1

Virtqueue 3 (RX)
  qsz 256 last_avail_idx 0 last_used_idx 0
  avail.flags 1 avail.idx 0 used.flags 1 used.idx 0
  kickfd 72 callfd 74 errfd -1

Virtqueue 4 (TX disabled)
  qsz 256 last_avail_idx 0 last_used_idx 0
  avail.flags 1 avail.idx 0 used.flags 1 used.idx 0
  kickfd 76 callfd 78 errfd -1

Virtqueue 5 (RX disabled)
  qsz 256 last_avail_idx 0 last_used_idx 0
  avail.flags 1 avail.idx 0 used.flags 1 used.idx 0
  kickfd 80 callfd 82 errfd -1

Virtqueue 6 (TX disabled)
  qsz 256 last_avail_idx 0 last_used_idx 0
  avail.flags 1 avail.idx 0 used.flags 1 used.idx 0
  kickfd 84 callfd 86 errfd -1

```

(continues on next page)

(continued from previous page)

```
Virtqueue 7 (RX disabled)
  qsz 256 last_avail_idx 0 last_used_idx 0
  avail.flags 1 avail.idx 0 used.flags 1 used.idx 0
  kickfd 88 callfd 90 errfd -1
```

The optional ‘*descriptors*’ parameter will display the same output as the previous example but will include the descriptor table for each queue. The output is truncated below:

```
vpp# show vhost-user VirtualEthernet0/0/0 descriptors

Virtio vhost-user interfaces
Global:
  coalesce frames 32 time 1e-3
Interface: VirtualEthernet0/0/0 (ifindex 1)
virtio_net_hdr_sz 12
features mask (0xffffffffffffffff):
features (0x50408000):
  VIRTIO_NET_F_MRG_RXBUF (15)
  VIRTIO_NET_F_MQ (22)
:
Virtqueue 0 (TX)
  qsz 256 last_avail_idx 0 last_used_idx 0
  avail.flags 1 avail.idx 128 used.flags 1 used.idx 0
  kickfd 62 callfd 64 errfd -1

descriptor table:
  id          addr          len  flags  next  user_addr
  =====
0      0x0000000010b6e974 2060 0x0002 1      0x00002aabb76e974
1      0x0000000010b6e034 2060 0x0002 2      0x00002aabb76e034
2      0x0000000010b6d6f4 2060 0x0002 3      0x00002aabb76d6f4
3      0x0000000010b6cdb4 2060 0x0002 4      0x00002aabb76cdb4
4      0x0000000010b6c474 2060 0x0002 5      0x00002aabb76c474
5      0x0000000010b6bb34 2060 0x0002 6      0x00002aabb76bb34
6      0x0000000010b6b1f4 2060 0x0002 7      0x00002aabb76b1f4
7      0x0000000010b6a8b4 2060 0x0002 8      0x00002aabb76a8b4
8      0x0000000010b69f74 2060 0x0002 9      0x00002aabb769f74
9      0x0000000010b69634 2060 0x0002 10     0x00002aabb769634
10     0x0000000010b68cf4 2060 0x0002 11     0x00002aabb768cf4
:
249    0x0000000000000000 0      0x0000 250    0x00002aab2b400000
250    0x0000000000000000 0      0x0000 251    0x00002aab2b400000
251    0x0000000000000000 0      0x0000 252    0x00002aab2b400000
252    0x0000000000000000 0      0x0000 253    0x00002aab2b400000
253    0x0000000000000000 0      0x0000 254    0x00002aab2b400000
254    0x0000000000000000 0      0x0000 255    0x00002aab2b400000
255    0x0000000000000000 0      0x0000 32768 0x00002aab2b400000

Virtqueue 1 (RX)
  qsz 256 last_avail_idx 0 last_used_idx 0
```

Debug Vhost-User

Turn on/off debug for vhost

Summary/Usage

```
debug vhost-user <on | off>.
```

Delete Vhost-User

Delete a vHost User interface using the interface name or the software interface index. Use the *'show interface'* command to determine the software interface index. On deletion, the linux socket will not be deleted.

Summary/Usage

```
delete vhost-user {<interface> | sw_if_index <sw_idx>}.
```

Examples

Example of how to delete a vhost interface by name:

```
vpp# delete vhost-user VirtualEthernet0/0/1
```

Example of how to delete a vhost interface by software interface index:

```
vpp# delete vhost-user sw_if_index 1
```