Open Virtual Network (OVN)

Release 22.06.90

Jul 05, 2022

Contents

CHAPTER 1

Open Virtual Network (OVN) Documentation

1.1 How the Documentation is Organised

The Open Virtual Network (OVN) documentation is organised into multiple sections:

- *Installation guides* guide you through installing Open vSwitch (OVS) and Open Virtual Network (OVN) on a variety of different platforms
- *Tutorials* take you through a series of steps to configure OVS and OVN in sandboxed environments
- *Topic guides* provide a high level overview of OVS and OVN internals and operation
- *How-to guides* are recipes or use-cases for OVS and OVN. They are more advanced than the tutorials.
- *Frequently Asked Questions* provide general insight into a variety of topics related to configuration and operation of OVS and OVN.

1.2 First Steps

Getting started with Open Virtual Network (OVN) for Open vSwitch? Start here.

- **Install:** *OVN on Linux, FreeBSD and NetBSD* | *OVN on Windows*
- **Tutorials:** *OVN Sandbox* | *OVN OpenStack Tutorial* | *OVN IPsec Tutorial* | *OVN Role-Based Access Control (RBAC) Tutorial*

1.3 Deeper Dive

- **Architecture** *Integration Guide for Centralized Control*
- **Testing** *Testing*

- **Packaging:** *Debian Packaging for OVN* | *RHEL 5.6, 6.x Packaging for Open vSwitch* | *Fedora, RHEL 7.x Packaging for OVN*

1.4 The Open Virtual Network (OVN) Project

Learn more about the Open Virtual Network (OVN) project and about how you can contribute:

- **Community:** *OVN Release Process* | *Authors* | *Mailing Lists* | *Patchwork* | *Reporting Bugs in OVN* | *OVN's Security Process*
- **Contributing:** *Submitting Patches* | *Backporting patches* | *OVN Coding Style*
- **Maintaining:** *Committers* | *Expectations for Developers with OVN Repo Access* | *OVN Committer Grant/Revocation Policy* | *Emeritus Status for OVN Committers*
- **Documentation:** *OVN Documentation Style* | *Building OVN Documentation* | *How OVN's Documentation Works*

1.5 Getting Help

- Seeing an issue of potential bug? Report problems to bugs@openvswitch.org
- Looking for specific information? Try the genindex, modindex or the *detailed table of contents*.

CHAPTER 2

Getting Started

How to get started with the Open Virtual Network (OVN).

2.1 Installing Open Virtual Network (OVN)

A collection of guides detailing how to install OVN in a variety of different environments and using different configurations.

2.1.1 Installation from Source

OVN on Linux, FreeBSD and NetBSD

This document describes how to build and install OVN on a generic Linux, FreeBSD, or NetBSD host. For specifics around installation on a specific platform, refer to one of the other installation guides listed in *Installing Open Virtual Network (OVN)*.

Obtaining OVN Sources

The canonical location for OVN source code is its Git repository, which you can clone into a directory named "ovn" with:

```
$ git clone https://github.com/ovn-org/ovn.git
```

Cloning the repository leaves the "master" branch initially checked out. This is the right branch for general development. If, on the other hand, if you want to build a particular released version, you can check it out by running a command such as the following from the "ovn" directory:

```
$ git checkout v20.09.0
```

The repository also has a branch for each release series. For example, to obtain the latest fixes in the OVN 20.09.x release series, which might include bug fixes that have not yet been in any released version, you can check it out from the "ovn" directory with:

```
$ git checkout origin/branch-20.09
```

If you do not want to use Git, you can also obtain tarballs for *OVN release versions <https://www.ovn.org/en/releases/>*, or download a ZIP file for any snapshot from the *GitHub web interface <https://github.com/ovn-org/ovn>*.

Build Requirements

To compile the userspace programs in the OVN distribution, you will need the following software:

- Open vSwitch (https://docs.openvswitch.org/en/latest/intro/install/). Open vSwitch is included as a submodule in the OVN source code. It is kept at the minimum recommended version for OVN to build and operate optimally. See below for instructions about how to use a different OVS source location.

  Note: These OVS sources used as a set of libraries to build OVN binaries, so OVS submodule is only recommended to build OVN and *not recommended* to be used as a source for OVS build. To actually build/run OVS binaries (`ovs-vswitchd`, `ovsdb-server`) use released versions of Open vSwitch or packages provided in your distribution.

- GNU make
- One of the following C compilers:
  - GCC 4.6 or later.
  - Clang 3.4 or later.
  - MSVC 2013. Refer to *OVN on Windows* for additional Windows build instructions.
- libssl, from OpenSSL, is optional but recommended if you plan to connect the OVN services to the OVN DB ovsdb-servers securely. If libssl is installed, then OVN will automatically build with support for it.
- Unbound library, from http://www.unbound.net, is optional but recommended if you want to enable ovn-northd, ovn-controller and other utilities to use DNS names when specifying OVSDB remotes. If unbound library is already installed, then OVN will automatically build with support for it. The environment variable OVS_RESOLV_CONF can be used to specify DNS server configuration file (the default file on Linux is /etc/resolv.conf).
- *DDlog <https://github.com/vmware/differential-datalog>*, if you want to build `ovn-northd-ddlog`, an alternate implementation of `ovn-northd` that scales better to large deployments. The NEWS file specifies the right version of DDlog to use with this release. Building with DDlog supports requires Rust to be installed (see https://www.rust-lang.org/tools/install).

If you are working from a Git tree or snapshot (instead of from a distribution tarball), or if you modify the OVN build system or the database schema, you will also need the following software:

- Autoconf version 2.63 or later.
- Automake version 1.10 or later.
- libtool version 2.4 or later. (Older versions might work too.)

The OVN manpages will include an E-R diagram, in formats other than plain text, only if you have the following:

- dot from graphviz (http://www.graphviz.org/).

If you are going to extensively modify OVN, consider installing the following to obtain better warnings:

- "sparse" version 0.5.1 or later (https://git.kernel.org/pub/scm/devel/sparse/sparse.git/).
- GNU make.
- clang, version 3.4 or later
- flake8 along with the hacking flake8 plugin (for Python code). The automatic flake8 check that runs against Python code has some warnings enabled that come from the "hacking" flake8 plugin. If it's not installed, the warnings just won't occur until it's run on a system with "hacking" installed.

You may find the ovs-dev script found in `ovs/utilities/ovs-dev.py` useful.

Installation Requirements

The machine you build OVN on may not be the one you run it on. To simply install and run OVN you require the following software:

- Shared libraries compatible with those used for the build.

On Linux you should ensure that `/dev/urandom` exists. To support TAP devices, you must also ensure that `/dev/net/tun` exists.

Bootstrapping

This step is not needed if you have downloaded a released tarball. If you pulled the sources directly from an OVN Git tree or got a Git tree snapshot, then run boot.sh in the top source directory to build the "configure" script:

```
$ ./boot.sh
```

Before configuring OVN, prepare Open vSwitch sources. The easiest way to do this is to use the included OVS submodule in the OVN source tree:

```
$ git submodule update --init
$ cd ovs
$ ./boot.sh
$ ./configure
$ make
$ cd ..
```

It is not required to build with the included OVS submodule; however the OVS submodule is guaranteed to include minimum recommended version of OVS libraries to ensure OVN's build and optimal operation. If you wish to build with OVS source code from a different location on the file system, then be sure to configure and build it before building OVN.

Configuring

Then configure the package by running the configure script:

```
$ ./configure
```

If your OVS source directory is not the included OVS submodule, specify the location of the OVS source code using –with-ovs-source:

```
$ ./configure --with-ovs-source=/path/to/ovs/source
```

If you have built Open vSwitch in a separate directory from its source code, then you need to provide that path in the option - –with-ovs-build.

By default all files are installed under `/usr/local`. OVN expects to find its database in `/usr/local/etc/ovn` by default. If you want to install all files into, e.g., `/usr` and `/var` instead of `/usr/local` and `/usr/local/var` and expect to use `/etc/ovn` as the default database directory, add options as shown here:

```
$ ./configure --prefix=/usr --localstatedir=/var --sysconfdir=/etc
```

Note: OVN installed with packages like .rpm (e.g. via `yum install` or `rpm -ivh`) and .deb (e.g. via `apt-get install` or `dpkg -i`) use the above configure options.

Use `--with-ddlog` to build with DDlog support. To build with DDlog, the build system needs to be able to find the `ddlog` and `ovsdb2ddlog` binaries and the DDlog library directory (the directory that contains `ddlog_std.dl`). This option supports a few ways to do that:

- If binaries are in $PATH, use the library directory as argument, e.g. `--with-ddlog=$HOME/differential-datalog/lib`. This is suitable if DDlog was installed from source via `stack install` or from (hypothetical) distribution packaging.

  The DDlog documentation recommends pointing $DDLOG_HOME to the DDlog source directory. If you did this, so that $DDLOG_HOME/lib is the library directory, you may use `--with-ddlog` without an argument.

- If the binaries and libraries are in the `bin` and `lib` subdirectories of an installation directory, use the installation directory as the argument. This is suitable if DDlog was installed from one of the binary tarballs published by the DDlog developers.

Note: Building with DDLog adds a few minutes to the build because the Rust compiler is slow. Add `--enable-ddlog-fast-build` to make this about 2x faster. This disables some Rust compiler optimizations, making a much slower `ovn-northd-ddlog` executable, so it should not be used for production builds or for profiling.

By default, static libraries are built and linked against. If you want to use shared libraries instead:

```
$ ./configure --enable-shared
```

To use a specific C compiler for compiling Open vSwitch user programs, also specify it on the configure command line, like so:

```
$ ./configure CC=gcc-4.2
```

To use ‘clang’ compiler:

```
$ ./configure CC=clang
```

To supply special flags to the C compiler, specify them as `CFLAGS` on the configure command line. If you want the default CFLAGS, which include `-g` to build debug symbols and `-O2` to enable optimizations, you must include them yourself. For example, to build with the default CFLAGS plus `-mssse3`, you might run configure as follows:

```
$ ./configure CFLAGS="-g -O2 -mssse3"
```

For efficient hash computation special flags can be passed to leverage built-in intrinsics. For example on X86_64 with SSE4.2 instruction set support, CRC32 intrinsics can be used by passing `-msse4.2`:

```
$ ./configure CFLAGS="-g -O2 -msse4.2"`
```

Also builtin popcnt instruction can be used to speedup the counting of the bits set in an integer. For example on X86_64 with POPCNT support, it can be enabled by passing `-mpopcnt`:

```
$ ./configure CFLAGS="-g -O2 -mpopcnt"`
```

If you are on a different processor and don't know what flags to choose, it is recommended to use `-march=native` settings:

```
$ ./configure CFLAGS="-g -O2 -march=native"
```

With this, GCC will detect the processor and automatically set appropriate flags for it. This should not be used if you are compiling OVS outside the target machine.

Note: CFLAGS are not applied when building the Linux kernel module. Custom CFLAGS for the kernel module are supplied using the `EXTRA_CFLAGS` variable when running make. For example:

```
$ make EXTRA_CFLAGS="-Wno-error=date-time"
```

If you are a developer and want to enable Address Sanitizer for debugging purposes, at about a 2x runtime cost, you can add `-fsanitize=address -fno-omit-frame-pointer -fno-common` to CFLAGS. For example:

```
$ ./configure CFLAGS="-g -O2 -fsanitize=address -fno-omit-frame-pointer -fno-common"
```

To build the Linux kernel module, so that you can run the kernel-based switch, pass the location of the kernel build directory on `--with-linux`. For example, to build for a running instance of Linux:

```
$ ./configure --with-linux=/lib/modules/$(uname -r)/build
```

Note: If `--with-linux` requests building for an unsupported version of Linux, then `configure` will fail with an error message. Refer to the *Open Virtual Network (OVN) FAQ* for advice in that case.

If you plan to do much OVN development, you might want to add `--enable-Werror`, which adds the `-Werror` option to the compiler command line, turning warnings into errors. That makes it impossible to miss warnings generated by the build. For example:

```
$ ./configure --enable-Werror
```

If you're building with GCC, then, for improved warnings, install `sparse` (see "Prerequisites") and enable it for the build by adding `--enable-sparse`. Use this with `--enable-Werror` to avoid missing both compiler and `sparse` warnings, e.g.:

```
$ ./configure --enable-Werror --enable-sparse
```

To build with gcov code coverage support, add `--enable-coverage`:

```
$ ./configure --enable-coverage
```

The configure script accepts a number of other options and honors additional environment variables. For a full list, invoke configure with the `--help` option:

```
$ ./configure --help
```

You can also run configure from a separate build directory. This is helpful if you want to build OVN in more than one way from a single source directory, e.g. to try out both GCC and Clang builds. For example:

```
$ mkdir _gcc && (cd _gcc && ./configure CC=gcc)
$ mkdir _clang && (cd _clang && ./configure CC=clang)
```

Under certain loads the ovsdb-server and other components perform better when using the jemalloc memory allocator, instead of the glibc memory allocator. If you wish to link with jemalloc add it to LIBS:

```
$ ./configure LIBS=-ljemalloc
```

Example usage:: $ # Clone OVS repo $cd /home/foo/ovs $./boot.sh $mkdir _gcc $cd _gcc && ../configure && cd .. $make -C _gcc

$ # Clone OVN repo $cd /home/foo/ovn $./boot.sh $./configure –with-ovs-source=/home/foo/ovs/ –with-ovs-build=/home/foo/ovs/_gcc

It is expected to configure both Open vSwitch and OVN with the same prefix.

Building

1. Run GNU make in the build directory, e.g.:

   ```
   $ make
   ```

   or if GNU make is installed as "gmake":

   ```
   $ gmake
   ```

   If you used a separate build directory, run make or gmake from that directory, e.g.:

   ```
   $ make -C _gcc
   $ make -C _clang
   ```

   Note: Some versions of Clang and ccache are not completely compatible. If you see unusual warnings when you use both together, consider disabling ccache.

2. Consider running the testsuite. Refer to *Testing* for instructions.

3. Run `make install` to install the executables and manpages into the running system, by default under `/usr/local`:

   ```
   $ sudo make install
   ```

Starting

Before starting the OVN, start the Open vSwitch daemons. Refer to the Open vSwitch documentation for more details on how to start OVS.

On Unix-alike systems, such as BSDs and Linux, starting the OVN suite of daemons is a simple process. OVN includes a shell script, called ovn-ctl which automates much of the tasks for starting and stopping ovn-northd, ovn-controller and

ovsdb-servers. After installation, the daemons can be started by using the ovn-ctl utility. This will take care to setup initial conditions, and start the daemons in the correct order. The ovn-ctl utility is located in '$(pkgdatadir)/scripts', and defaults to '/usr/local/share/ovn/scripts'. ovn-ctl utility requires the 'ovs-lib' helper shell script which is present in '/usr/local/share/openvswitch/scripts'. So invoking ovn-ctl as "./ovn-ctl" will fail.

An example after install might be:

```
$ export PATH=$PATH:/usr/local/share/ovn/scripts
$ ovn-ctl start_northd
$ ovn-ctl start_controller
```

If you built with DDlog support, then you can start `ovn-northd-ddlog` instead of `ovn-northd` by adding `--ovn-northd-ddlog=yes`, e.g.:

```
$ export PATH=$PATH:/usr/local/share/ovn/scripts
$ ovn-ctl --ovn-northd-ddlog=yes start_northd
$ ovn-ctl start_controller
```

Starting OVN Central services

OVN central services includes ovn-northd, Northbound and Southbound ovsdb-server.

$ export PATH=$PATH:/usr/local/share/ovn/scripts $ ovn-ctl start_northd

Refer to ovn-ctl(8) for more information and the supported options.

You may wish to manually start the OVN central daemons. Before starting ovn-northd you need to start OVN Northbound and Southbound ovsdb-servers. Before ovsdb-servers can be started, configure the Northbound and Southbound databases:

```
$ sudo mkdir -p /usr/local/etc/ovn
$ sudo ovsdb-tool create /usr/local/etc/ovn/ovnnb_db.db \
  ovn-nb.ovsschema
$ sudo ovsdb-tool create /usr/local/etc/ovn/ovnsb_db.db \
  ovn-sb.ovsschema
```

Configure ovsdb-servers to use databases created above, to listen on a Unix domain socket and to use the SSL configuration in the database:

```
$ sudo mkdir -p /usr/local/var/run/ovn
$ sudo ovsdb-server /usr/local/etc/ovn/ovnnb_db.db --remote=punix:/usr/local/var/run/
↪ovn/ovnnb_db.sock \
     --remote=db:OVN_Northbound,NB_Global,connections \
     --private-key=db:OVN_Northbound,SSL,private_key \
     --certificate=db:OVN_Northbound,SSL,certificate \
     --bootstrap-ca-cert=db:OVN_Northbound,SSL,ca_cert \
     --pidfile=/usr/local/var/run/ovn/ovnnb-server.pid --detach --log-file=/usr/local/
↪var/log/ovn/ovnnb-server.log
$ sudo ovsdb-server /usr/local/etc/ovn/ovnsb_db.db --remote=punix:/usr/local/var/run/
↪ovn/ovnsb_db.sock \
     --remote=db:OVN_Southbound,SB_Global,connections \
     --private-key=db:OVN_Southbound,SSL,private_key \
     --certificate=db:OVN_Southbound,SSL,certificate \
     --bootstrap-ca-cert=db:OVN_Southbound,SSL,ca_cert \
     --pidfile=/usr/local/var/run/ovn/ovnsb-server.pid --detach --log-file=/usr/local/
↪var/log/ovn/ovnsb-server.log
```

Note: If you built OVN without SSL support, then omit `--private-key`, `--certificate`, and `--bootstrap-ca-cert`.)

Initialize the databases using ovn-nbctl and ovn-sbctl. This is only necessary the first time after you create the databases with ovsdb-tool, though running it at any time is harmless:

```
$ ovn-nbctl --no-wait init
$ ovn-sbctl --no-wait init
```

Start `ovn-northd`, telling it to connect to the OVN db servers same Unix domain socket:

```
$ ovn-northd --pidfile --detach --log-file
```

If you built with DDlog support, you can start `ovn-northd-ddlog` instead, the same way:

```
$ ovn-northd-ddlog --pidfile --detach --log-file
```

Starting OVN Central services in containers

For OVN central node, we dont need to load ovs kernel modules on host. Hence, OVN central containers OS need not depend on host OS.

Also we can leverage deploying entire OVN control plane in a pod spec for use cases like OVN-kubernetes

Export following variables in .env and place it under project root:

```
$ OVN_BRANCH=<BRANCH>
$ OVN_VERSION=<VERSION>
$ DISTRO=<LINUX_DISTRO>
$ KERNEL_VERSION=<LINUX_KERNEL_VERSION>
$ GITHUB_SRC=<GITHUB_URL>
$ DOCKER_REPO=<REPO_TO_PUSH_IMAGE>
```

To build ovn modules:

```
$ cd utilities/docker
$ make build
```

Compiled Modules will be tagged with docker image

To Push ovn modules:

```
$ make push
```

OVN docker image will be pushed to specified docker repo.

Start OVN containers using below command:

```
$ docker run -itd --net=host --name=ovn-nb \
  <docker_repo>:<tag> ovn-nb-tcp

$ docker run -itd --net=host --name=ovn-sb \
  <docker_repo>:<tag> ovn-sb-tcp

$ docker run -itd --net=host --name=ovn-northd \
  <docker_repo>:<tag> ovn-northd-tcp
```

Start OVN containers in cluster mode for a 3 node cluster using below command on node1:

```
$ docker run -e "host_ip=<host_ip>" -e "nb_db_port=<port>" -itd \
  --name=ovn-nb-raft --net=host --privileged <docker_repo>:<tag> \
  ovn-nb-cluster-create

$ docker run -e "host_ip=<host_ip>" -e "sb_db_port=<port>" -itd \
  --name=ovn-sb-raft --net=host --privileged <docker_repo>:<tag> \
  ovn-sb-cluster-create

$ docker run -e "OVN_NB_DB=tcp:<node1>:6641,tcp:<node2>:6641,\
  tcp:<node3>:6641" -e "OVN_SB_DB=tcp:<node1>:6642,tcp:<node2>:6642,\
  tcp:<node3>:6642" -itd --name=ovn-northd-raft <docker_repo>:<tag> \
  ovn-northd-cluster
```

Start OVN containers in cluster mode using below command on node2 and node3 to make them join the peer using below command:

```
$ docker run -e "host_ip=<host_ip>" -e "remote_host=<remote_host_ip>" \
  -e "nb_db_port=<port>" -itd --name=ovn-nb-raft --net=host \
  --privileged <docker_repo>:<tag> ovn-nb-cluster-join

$ docker run -e "host_ip=<host_ip>" -e "remote_host=<remote_host_ip>" \
  -e "sb_db_port=<port>" -itd --name=ovn-sb-raft --net=host \
  --privileged <docker_repo>:<tag> ovn-sb-cluster-join

$ docker run -e "OVN_NB_DB=tcp:<node1>:6641,tcp:<node2>:6641,\
  tcp:<node3>:6641" -e "OVN_SB_DB=tcp:<node1>:6642,tcp:<node2>:6642,\
  tcp:<node3>:6642" -itd --name=ovn-northd-raft <docker_repo>:<tag> \
  ovn-northd-cluster
```

Start OVN containers using unix socket:

```
$ docker run -itd --net=host --name=ovn-nb \
  -v /var/run/ovn/:/var/run/ovn/ \
  <docker_repo>:<tag> ovn-nb

$ docker run -itd --net=host --name=ovn-sb \
  -v /var/run/ovn/:/var/run/ovn/
  <docker_repo>:<tag> ovn-sb

$ docker run -itd --net=host --name=ovn-northd \
  -v /var/run/ovn/:/var/run/ovn/
  <docker_repo>:<tag> ovn-northd
```

Note: Current ovn central components comes up in docker image in a standalone and cluster mode with protocol tcp.

The debian docker file use ubuntu 16.04 as a base image for reference.

User can use any other base image for debian, e.g. u14.04, etc.

RHEL based docker support is now added with centos7 as a base image.

Starting OVN host service

On each chassis, ovn-controller service should be started. ovn-controller assumes it gets configuration information from the Open_vSwitch table of the local OVS instance. Refer to the ovn-controller(8) for the configuration keys.

Below are the required keys to be configured on each chassis.

1. external_ids:system-id
2. external_ids:ovn-remote
3. external_ids:ovn-encap-type
4. external_ids:ovn-encap-ip

You may wish to manually start the ovn-controller service on each chassis.

Start the ovn-controller, telling it to connect to the local ovsdb-server Unix domain socket:

```
$ ovn-controller --pidfile --detach --log-file
```

Starting OVN host service in containers

For OVN host too, we dont need to load ovs kernel modules on host. Hence, OVN host container OS need not depend on host OS.

Also we can leverage deploying OVN host in a pod spec for use cases like OVN-kubernetes to manage OVS which can be running as a service on host or in container.

Start ovsdb-server and ovs-vswitchd components as per http://docs.openvswitch.org/en/latest/intro/install/general/

start local ovn-controller with below command if ovs is also running in container:

```
$ docker run -itd --net=host --name=ovn-controller \
  --volumes-from=ovsdb-server \
  <docker_repo>:<tag> ovn-controller
```

start local ovn-controller with below command if ovs is running as a service:

```
$ docker run -itd --net=host --name=ovn-controller \
  -v /var/run/openvswitch/:/var/run/openvswitch/ \
  <docker_repo>:<tag> ovn-controller
```

Validating

At this point you can use ovn-nbctl on the central node to set up logical switches and ports and other OVN logical entities. For example, to create a logical switch `sw0` and add logical port `sw0-p1`

```
$ ovn-nbctl ls-add sw0
$ ovn-nbctl lsp-add sw0 sw0-p1
$ ovn-nbctl show
```

Refer to ovn-nbctl(8) and ovn-sbctl (8) for more details.

When using ovn in container, exec to container to run above commands:

```
$ docker exec -it <ovn-nb/ovn-sb/ovn-northd/ovn-controller> /bin/bash
```

Reporting Bugs

Report problems to bugs@openvswitch.org.

OVN on Windows

TODO

- This document needs to be updated.

2.1.2 Installation from Packages

OVN is packaged on a variety of distributions. The tooling required to build these packages is included in the OVN tree. The instructions are provided below.

Distributions packaging Open Virtual Network (OVN)

This document lists various popular distributions packaging OVN.

Note: The packaged version available with distributions may not be latest OVN release.

Debian

You can use `apt-get` or `aptitude` to install the .deb packages and must be superuser. Debian has `ovn-common`, `ovn-host`, `ovn-central` and `ovn-vtep` .deb packages.

Fedora

Fedora provides `ovn`, `ovn-host`, `ovn-central` and `ovn-vtep` rpm packages. Use `yum` or `dnf` to install the rpm packages and must be superuser.

OpenSuSE

OpenSUSE provides `openvswitch-ovn-common`, `` `openvswitch-ovn-host ``, `` `openvswitch-ovn-central `` and `` `openvswitch-ovn-vtep `` rpm packages.

Debian Packaging for OVN

OVN packages needs to be split from OVS. This section will be updated once it is done.

Fedora, RHEL 7.x Packaging for OVN

This document provides instructions for building and installing OVN RPM packages on a Fedora Linux host. Instructions for the installation of OVN Fedora Linux host without using RPM packages can be found in the *OVN on Linux, FreeBSD and NetBSD*.

These instructions have been tested with Fedora 29 and 30, and are also applicable for RHEL 7.x and its derivatives, including CentOS 7.x and Scientific Linux 7.x.

Build Requirements

You will need to install all required packages to build the RPMs. Newer distributions use `dnf` but if it's not available, then use `yum` instructions.

The command below will install RPM tools and generic build dependencies. And (optionally) include these packages: libcap-ng libcap-ng-devel.

DNF:

```
$ dnf install @'Development Tools' rpm-build dnf-plugins-core
```

YUM:

```
$ yum install @'Development Tools' rpm-build yum-utils
```

Then it is necessary to install OVN specific build dependencies. The dependencies are listed in the SPEC file, but first it is necessary to replace the VERSION tag to be a valid SPEC.

The command below will create a temporary SPEC file:

```
$ sed -e 's/@VERSION@/0.0.1/' rhel/ovn-fedora.spec.in \
  > /tmp/ovn.spec
```

And to install specific dependencies, use the corresponding tool below. For some of the dependencies on RHEL you may need to add two additional repositories to help yum-builddep, e.g.:

```
$ subscription-manager repos --enable=rhel-7-server-extras-rpms
$ subscription-manager repos --enable=rhel-7-server-optional-rpms
```

DNF:

```
$ dnf builddep /tmp/ovn.spec
```

YUM:

```
$ yum-builddep /tmp/ovn.spec
```

Once that is completed, remove the file `/tmp/ovn.spec`.

Bootstraping

Refer to *Bootstrapping*.

Configuring

Refer to *Configuring*.

Building

OVN RPMs

To build OVN RPMs, first generate openvswitch source tarball in your openvwitch source directory by running

```
$make dist
```

And then execute the following in the OVN source directory (in which *./configure* was executed):

```
$ make rpm-fedora
```

This will create the RPMs *ovn*, *ovn-central*, *ovn-host*, *ovn-vtep*, *ovn-docker*, `ovn-debuginfo`, `ovn-central-debuginfo`, `ovn-host-debuginfo` and `` `ovn-vtep-debuginfo` ``.

You can also have the above commands automatically run the OVN unit tests. This can take several minutes.

```
$ make rpm-fedora RPMBUILD_OPT="--with check"
```

Installing

RPM packages can be installed by using the command `rpm -i`. Package installation requires superuser privileges.

Refer to the RHEL README for additional usage and configuration information.

Reporting Bugs

Report problems to bugs@openvswitch.org.

RHEL 5.6, 6.x Packaging for Open vSwitch

This document describes how to build and install Open vSwitch on a Red Hat Enterprise Linux (RHEL) host. If you want to install Open vSwitch on a generic Linux host, refer to *OVN on Linux, FreeBSD and NetBSD* instead.

We have tested these instructions with RHEL 5.6 and RHEL 6.0.

For RHEL 7.x (or derivatives, such as CentOS 7.x), you should follow the instructions in the *Fedora, RHEL 7.x Packaging for OVN*. The Fedora spec files are used for RHEL 7.x.

Prerequisites

You may build from an Open vSwitch distribution tarball or from an Open vSwitch Git tree.

The default RPM build directory, `_topdir`, has five directories in the top-level.

BUILD/ where the software is unpacked and built

RPMS/ where the newly created binary package files are written

SOURCES/ contains the original sources, patches, and icon files

SPECS/ contains the spec files for each package to be built

SRPMS/ where the newly created source package files are written

Before you begin, note the RPM sources directory on your version of RHEL. The command `rpmbuild --showrc` will show the configuration for each of those directories. Alternatively, the command `rpm --eval '%{_topdir}'` shows the current configuration for the top level directory and the command `rpm --eval '%{_sourcedir}'` does the same for the sources directory. On RHEL 5, the default RPM `_topdir` is `/usr/src/redhat` and the default RPM sources directory is `/usr/src/redhat/SOURCES`. On RHEL 6, the default `_topdir` is `$HOME/rpmbuild` and the default RPM sources directory is `$HOME/rpmbuild/SOURCES`.

Build Requirements

You will need to install all required packages to build the RPMs. The command below will install RPM tools and generic build dependencies:

```
$ yum install @'Development Tools' rpm-build yum-utils
```

Then it is necessary to install Open vSwitch specific build dependencies. The dependencies are listed in the SPEC file, but first it is necessary to replace the VERSION tag to be a valid SPEC.

The command below will create a temporary SPEC file:

```
$ sed -e 's/@VERSION@/0.0.1/' rhel/openvswitch.spec.in > /tmp/ovs.spec
```

And to install specific dependencies, use yum-builddep tool:

```
$ yum-builddep /tmp/ovs.spec
```

Once that is completed, remove the file `/tmp/ovs.spec`.

If python-sphinx package is not available in your version of RHEL, you can install it via pip with 'pip install sphinx'.

Open vSwitch requires python 2.7 or newer which is not available in older distributions. In the case of RHEL 6.x and its derivatives, one option is to install python34 from EPEL.

Bootstrapping and Configuring

If you are building from a distribution tarball, skip to *Building*. If not, you must be building from an Open vSwitch Git tree. Determine what version of Autoconf is installed (e.g. run `autoconf --version`). If it is not at least version 2.63, then you must upgrade or use another machine to build the packages.

Assuming all requirements have been met, build the tarball by running:

```
$ ./boot.sh
$ ./configure
$ make dist
```

You must run this on a machine that has the tools listed in *Build Requirements* as prerequisites for building from a Git tree. Afterward, proceed with the rest of the instructions using the distribution tarball.

Now you have a distribution tarball, named something like `openvswitch-x.y.z.tar.gz`. Copy this file into the RPM sources directory, e.g.:

```
$ cp openvswitch-x.y.z.tar.gz $HOME/rpmbuild/SOURCES
```

Broken `build` symlink

Some versions of the RHEL 6 kernel-devel package contain a broken `build` symlink. If you are using such a version, you must fix the problem before continuing.

To find out whether you are affected, run:

```
$ cd /lib/modules/<version>
$ ls -l build/
```

where `<version>` is the version number of the RHEL 6 kernel.

Note: The trailing slash in the final command is important. Be sure to include it.

If the `ls` command produces a directory listing, your kernel-devel package is OK. If it produces a `No such file or directory` error, your kernel-devel package is buggy.

If your kernel-devel package is buggy, then you can fix it with:

```
$ cd /lib/modules/<version>
$ rm build
$ ln -s /usr/src/kernels/<target> build
```

where `<target>` is the name of an existing directory under `/usr/src/kernels`, whose name should be similar to `<version>` but may contain some extra parts. Once you have done this, verify the fix with the same procedure you used above to check for the problem.

Building

You should have a distribution tarball named something like openvswitch-x.y.z.tar.gz. Copy this file into the RPM sources directory:

```
$ cp openvswitch-x.y.z.tar.gz $HOME/rpmbuild/SOURCES
```

Make another copy of the distribution tarball in a temporary directory. Then unpack the tarball and `cd` into its root:

```
$ tar xzf openvswitch-x.y.z.tar.gz
$ cd openvswitch-x.y.z
```

Userspace

To build Open vSwitch userspace, run:

```
$ rpmbuild -bb rhel/openvswitch.spec
```

This produces two RPMs: “openvswitch” and “openvswitch-debuginfo”.

The above command automatically runs the Open vSwitch unit tests. To disable the unit tests, run:

```
$ rpmbuild -bb --without check rhel/openvswitch.spec
```

Note: If the build fails with `configure: error: source dir /lib/modules/2.6.32-279.el6.x86_64/build doesn't exist` or similar, then the kernel-devel package is missing or buggy.

Kernel Module

On RHEL 6, to build the Open vSwitch kernel module run:

```
$ rpmbuild -bb rhel/kmod-openvswitch-rhel6.spec
```

You might have to specify a kernel version and/or variants, e.g.:

> **$ rpmbuild -bb** -D "kversion 2.6.32-131.6.1.el6.x86_64" -D "kflavors default debug kdump" rhel/kmod-openvswitch-rhel6.spec

This produces an "kmod-openvswitch" RPM for each kernel variant, in this example: "kmod-openvswitch", "kmod-openvswitch-debug", and "kmod-openvswitch-kdump".

Red Hat Network Scripts Integration

A RHEL host has default firewall rules that prevent any Open vSwitch tunnel traffic from passing through. If a user configures Open vSwitch tunnels like Geneve, GRE, VXLAN, LISP etc., they will either have to manually add iptables firewall rules to allow the tunnel traffic or add it through a startup script Refer to the "enable-protocol" command in the ovs-ctl(8) manpage for more information.

In addition, simple integration with Red Hat network scripts has been implemented. Refer to README.RHEL.rst in the source tree or /usr/share/doc/openvswitch/README.RHEL.rst in the installed openvswitch package for details.

Reporting Bugs

Report problems to bugs@openvswitch.org.

2.1.3 Upgrades

OVN Upgrades

Since OVN is a distributed system, special consideration must be given to the process used to upgrade OVN across a deployment. This document discusses the recommended upgrade process.

Release Notes

You should always check the OVS and OVN release notes (NEWS file) for any release specific notes on upgrades.

OVS

OVN depends on and is included with OVS. It's expected that OVS and OVN are upgraded together, partly for convenience. OVN is included in OVS releases so it's easiest to upgrade them together. OVN may also make use of new features of OVS only available in that release.

Upgrade ovn-controller

You should start by upgrading ovn-controller on each host it's running on. First, you upgrade the OVS and OVN packages. Then, restart the ovn-controller service. You can restart with ovn-ctl:

```
$ sudo /usr/share/openvswitch/scripts/ovn-ctl restart_controller
```

or with systemd:

```
$ sudo systemd restart ovn-controller
```

Upgrade OVN Databases and ovn-northd

The OVN databases and ovn-northd should be upgraded next. Since ovn-controller has already been upgraded, it will be ready to operate on any new functionality specified by the database or logical flows created by ovn-northd.

Upgrading the OVN packages installs everything needed for an upgrade. The only step required after upgrading the packages is to restart ovn-northd, which automatically restarts the databases and upgrades the database schema, as well.

You may perform this restart using the ovn-ctl script:

```
$ sudo /usr/share/openvswitch/scripts/ovn-ctl restart_northd
```

or if you're using a Linux distribution with systemd:

```
$ sudo systemctl restart ovn-northd
```

Schema Change

During database upgrading, if there is schema change, the DB file will be converted to the new schema automatically, if the schema change is backward compatible. OVN tries the best to keep the DB schemas backward compatible.

However, there can be situations that an incompatible change is reasonble. An example of such case is to add constraints in the table to ensure correctness. If there were already data that violates the new constraints got added somehow, it will result in DB upgrade failures. In this case, user should manually correct data using ovn-nbctl (for north-bound DB) or ovn-sbctl (for south- bound DB), and then upgrade again following previous steps. Below is a list of known impactible schema changes and how to fix when error encountered.

1. Release 2.11: index [type, ip] added for Encap table of south-bound DB to prevent duplicated IPs being used for same tunnel type. If there are duplicated data added already (e.g. due to improper chassis management), a convenient way to fix is to find the chassis that is using the IP with command:

   ```
   $ ovn-sbctl show
   ```

   Then delete the chassis with command:

```
$ ovn-sbctl chassis-del <chassis>
```

Upgrading OVN Integration

Lastly, you may also want to upgrade integration with OVN that you may be using. For example, this could be the OpenStack Neutron driver or ovn-kubernetes.

OVN's northbound database schema is a backwards compatible interface, so you should be able to safely complete an OVN upgrade before upgrading any integration in use.

2.1.4 Others

Open Virtual Network (OVN) Documentation

This document describes how to build the OVN documentation for use offline. A continuously updated, online version can be found at docs.ovn.org.

Note: These instructions provide information on building the documentation locally. For information on writing documentation, refer to *OVN Documentation Style*

Build Requirements

As described in the *OVN Documentation Style*, the OVN documentation is written in reStructuredText and built with Sphinx. A detailed guide on installing Sphinx in many environments is available on the Sphinx website but, for most Linux distributions, you can install with your package manager. For example, on Debian/Ubuntu run:

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

Similarly, on RHEL/Fedora run:

```
$ sudo dnf install python-sphinx
```

A `requirements.txt` is also provided in the `/Documentation`, should you wish to install using `pip`:

```
$ virtualenv .venv
$ source .venv/bin/activate
$ pip install -r Documentation/requirements.txt
```

Configuring

It's unlikely that you'll need to customize any aspect of the configuration. However, the `Documentation/conf.py` is the go-to place for all configuration. This file is well documented and further information is available on the Sphinx website.

Building

Once Sphinx installed, the documentation can be built using the provided Makefile targets:

```
$ make docs-check
```

Important: The `docs-check` target will fail if there are any syntax errors. However, it won't catch more succint issues such as style or grammar issues. As a result, you should always inspect changes visually to ensure the result is as intended.

Once built, documentation is available in the `/Documentation/_build` folder. Open the root `index.html` to browse the documentation.

CHAPTER 3

Tutorials

Getting started with Open vSwitch (OVS) and Open Virtual Network (OVN) for Open vSwitch.

3.1 OVN Sandbox

This tutorial shows you how to explore features using `ovs-sandbox` as a simulated test environment. It's assumed that you have an understanding of OVS before going through this tutorial. Detail about OVN is covered in ovn-architecture, but this tutorial lets you quickly see it in action.

3.1.1 Getting Started

For some general information about `ovs-sandbox`, see the Open vSwitch documentaion on `ovs-sandbox`.

`ovs-sandbox` in the OVN repo includes OVN support by default. To start it, you would simply need to run:

```
$ make sandbox
```

Running the sandbox does the following steps to the environment:

1. Creates the `OVN_Northbound` and `OVN_Southbound` databases as described in ovn-nb(5) and ovn-sb(5).
2. Creates a backup server for `OVN_Southbond` database. Sandbox launch screen provides the instructions on accessing the backup database. However access to the backup server is not required to go through the tutorial.
3. Creates the `hardware_vtep` database as described in vtep(5).
4. Runs the ovn-northd(8), ovn-controller(8), and ovn-controller-vtep(8) daemons.
5. Makes OVN and VTEP utilities available for use in the environment, including vtep-ctl(8), ovn-nbctl(8), and ovn-sbctl(8).

3.1.2 Using GDB

GDB support is not required to go through the tutorial. See the "Using GDB" section of ovs-advanced in Open vSwitch documentation for more info. Additional flags exist for launching the debugger for the OVN programs:

```
--gdb-ovn-northd
--gdb-ovn-controller
--gdb-ovn-controller-vtep
```

3.1.3 Creating OVN Resources

Once you have `ovs-sandbox` running with OVN enabled, you can start using OVN utilities to create resources in OVN. As an example, we will create an environment that has two logical switches connected by a logical router.

Create the first logical switch with one port:

```
$ ovn-nbctl ls-add sw0
$ ovn-nbctl lsp-add sw0 sw0-port1
$ ovn-nbctl lsp-set-addresses sw0-port1 "50:54:00:00:00:01 192.168.0.2"
```

Create the second logical switch with one port:

```
$ ovn-nbctl ls-add sw1
$ ovn-nbctl lsp-add sw1 sw1-port1
$ ovn-nbctl lsp-set-addresses sw1-port1 "50:54:00:00:00:03 11.0.0.2"
```

Create the logical router and attach both logical switches:

```
$ ovn-nbctl lr-add lr0
$ ovn-nbctl lrp-add lr0 lrp0 00:00:00:00:ff:01 192.168.0.1/24
$ ovn-nbctl lsp-add sw0 lrp0-attachment
$ ovn-nbctl lsp-set-type lrp0-attachment router
$ ovn-nbctl lsp-set-addresses lrp0-attachment 00:00:00:00:ff:01
$ ovn-nbctl lsp-set-options lrp0-attachment router-port=lrp0
$ ovn-nbctl lrp-add lr0 lrp1 00:00:00:00:ff:02 11.0.0.1/24
$ ovn-nbctl lsp-add sw1 lrp1-attachment
$ ovn-nbctl lsp-set-type lrp1-attachment router
$ ovn-nbctl lsp-set-addresses lrp1-attachment 00:00:00:00:ff:02
$ ovn-nbctl lsp-set-options lrp1-attachment router-port=lrp1
```

View a summary of OVN's current logical configuration:

```
$ ovn-nbctl show
    switch 1396cf55-d176-4082-9a55-1c06cef626e4 (sw1)
        port lrp1-attachment
            addresses: ["00:00:00:00:ff:02"]
        port sw1-port1
            addresses: ["50:54:00:00:00:03 11.0.0.2"]
    switch 2c9d6d03-09fc-4e32-8da6-305f129b0d53 (sw0)
        port lrp0-attachment
            addresses: ["00:00:00:00:ff:01"]
        port sw0-port1
            addresses: ["50:54:00:00:00:01 192.168.0.2"]
    router f8377e8c-f75e-4fc8-8751-f3ea03c6dd98 (lr0)
        port lrp0
            mac: "00:00:00:00:ff:01"
```

(continues on next page)

(continued from previous page)

```
            networks: ["192.168.0.1/24"]
        port lrp1
            mac: "00:00:00:00:ff:02"
            networks: ["11.0.0.1/24"]
```

The `tutorial` directory of the OVS source tree includes a script that runs all of the commands for you:

```
$ ./ovn-setup.sh
```

3.1.4 Using ovn-trace

Once you have configured resources in OVN, try using `ovn-trace` to see how OVN would process a sample packet through its logical pipeline.

For example, we can trace an IP packet from `sw0-port1` to `sw1-port1`. The `--minimal` output shows each visible action performed on the packet, which includes:

1. The logical router will decrement the IP TTL field.
2. The logical router will change the source and destination MAC addresses to reflect the next hop.
3. The packet will be output to `sw1-port1`.

```
$ ovn-trace --minimal sw0 'inport == "sw0-port1" \
> && eth.src == 50:54:00:00:00:01 && ip4.src == 192.168.0.2 \
> && eth.dst == 00:00:00:00:ff:01 && ip4.dst == 11.0.0.2 \
> && ip.ttl == 64'

# ip,reg14=0x1,vlan_tci=0x0000,dl_src=50:54:00:00:00:01,dl_dst=00:00:00:00:ff:01,nw_
↪src=192.168.0.2,nw_dst=11.0.0.2,nw_proto=0,nw_tos=0,nw_ecn=0,nw_ttl=64
ip.ttl--;
eth.src = 00:00:00:00:ff:02;
eth.dst = 50:54:00:00:00:03;
output("sw1-port1");
```

The `ovn-trace` utility can also provide much more detail on how the packet would be processed through OVN's logical pipeline, as well as correlate that to OpenFlow flows programmed by `ovn-controller`. See the ovn-trace(8) man page for more detail.

3.2 OVN OpenStack Tutorial

This tutorial demonstrates how OVN works in an OpenStack "DevStack" environment. It was tested with the "master" branches of DevStack and Open vSwitch near the beginning of May 2017. Anyone using an earlier version is likely to encounter some differences. In particular, we noticed some shortcomings in OVN utilities while writing the tutorial and pushed out some improvements, so it's best to use recent Open vSwitch at least from that point of view.

The goal of this tutorial is to demonstrate OVN in an end-to-end way, that is, to show how it works from the cloud management system at the top (in this case, OpenStack and specifically its Neutron networking subsystem), through the OVN northbound and southbound databases, to the bottom at the OVN local controller and Open vSwitch data plane. We hope that this demonstration makes it easier for users and potential users to understand how OVN works and how to debug and troubleshoot it.

In addition to new material, this tutorial incorporates content from `ovn_devstack.rst` in OpenStack neutron, by Russell Bryant and others. Without that example, this tutorial could not have been written.

We provide enough details in the tutorial that you should be able to fully follow along, by creating a DevStack VM and cloning DevStack and so on. If you want to do this, start out from *Setting Up DevStack* below.

3.2.1 Setting Up DevStack

This section explains how to install DevStack, a kind of OpenStack packaging for developers, in a way that allows you to follow along with the tutorial in full.

Unless you have a spare computer laying about, it's easiest to install DevStack in a virtual machine. This tutorial was built using a VM implemented by KVM and managed by virt-manager. I recommend configuring the VM configured for the x86-64 architecture, 6 GB RAM, 2 VCPUs, and a 20 GB virtual disk.

Note: Since we will be creating VMs inside this VM, it is important to have nesting configured properly. See https://github.com/openstack/devstack/blob/master/doc/source/guides/devstack-with-nested-kvm.rst for details on that.

Also, if you happen to run your Linux-based host with 32-bit userspace, then you will have some special issues, even if you use a 64-bit kernel:

- You may find that you can get 32-bit DevStack VMs to work to some extent, but I personally got tired of finding workarounds. I recommend running your VMs in 64-bit mode. To get this to work, I had to go to the CPUs tab for the VM configuration in virt-manager and change the CPU model from the one originally listed to "Hypervisor Default' (it is curious that this is not the default!).
- On a host with 32-bit userspace, KVM supports VMs with at most 2047 MB RAM. This is adequate, barely, to start DevStack, but it is not enough to run multiple (nested) VMs. To prevent out-of-memory failures, set up extra swap space in the guest. For example, to add 2 GB swap:

  ```
  $ sudo dd if=/dev/zero of=/swapfile bs=1M count=2048
  $ sudo mkswap /swapfile
  $ sudo swapon /swapfile
  ```

  and then add a line like this to `/etc/fstab` to add the new swap automatically upon reboot:

  ```
  /swapfile swap swap defaults 0 0
  ```

Here are step-by-step instructions to get started:

1. Install a VM.

   I tested these instructions with Centos 7.6. Download the "minimal install" ISO and booted it. The install is straightforward. Be sure to enable networking, and set a host name, such as "ovn-devstack-1". Add a regular (non-root) user, and check the box "Make this user administrator". Also, set your time zone.

2. You can SSH into the DevStack VM, instead of running from a console. I recommend it because it's easier to cut and paste commands into a terminal than a VM console. You might also consider using a very wide terminal, perhaps 160 columns, to keep tables from wrapping.

   To improve convenience further, you can make it easier to log in with the following steps, which are optional:

   a. On your host, edit your `~/.ssh/config`, adding lines like the following:

      ```
      Host ovn-devstack-1
            Hostname VMIP
            User VMUSER
      ```

      where VMIP is the VM's IP address and VMUSER is your username inside the VM. (You can omit the `User` line if your username is the same in the host and the VM.) After you do this, you can SSH to the VM

by name, e.g. `ssh ovn-devstack-1`, and if command-line completion is set up in your host shell, you can shorten that to something like `ssh ovn` followed by hitting the Tab key.

b. If you have SSH public key authentication set up, with an SSH agent, run on your host:

```
$ ssh-copy-id ovn-devstack-1
```

and type your password once. Afterward, you can log in without typing your password again.

(If you don't already use SSH public key authentication and an agent, consider looking into it–it will save you time in the long run.)

c. Optionally, inside the VM, append the following to your `~/.bash_profile`:

```
. $HOME/devstack/openrc admin
```

It will save you running it by hand each time you log in. But it also prints garbage to the console, which can screw up services like `ssh-copy-id`, so be careful.

2. Boot into the installed system and log in as the regular user, then install Git:

```
$ sudo yum install git
```

Note: Support for Centos 7 in Devstack is going away, but you can still use it. Especially while Centos 8 support is not finished. The one important caveat for making Centos 7 work with Devstack is that you will explicitly have to install these packages as well:

```
$ sudo yum install python3 python3-devel
```

If you installed a 32-bit i386 guest (against the advice above), install a non-PAE kernel and reboot into it at this point:

```
$ sudo yum install kernel-core kernel-devel
$ sudo reboot
```

Be sure to select the non-PAE kernel from the list at boot. Without this step, DevStack will fail to install properly later.

---

3. Get copies of DevStack and Neutron and set them up:

```
$ git clone https://git.openstack.org/openstack-dev/devstack.git
$ git clone https://git.openstack.org/openstack/neutron.git
$ cd devstack
$ cp ../neutron/devstack/ovn-local.conf.sample local.conf
```

Note: Depending on the name of the network device used by the VM, devstack may be unable to automatically obtain its IP address. If that happens, edit `local.conf` and explicitly provide it (X marks the spot):

```
HOST_IP=X
```

If you installed a 32-bit i386 guest (against the advice above), at this point edit `local.conf` to add the following line:

```
CIRROS_ARCH=i386
```

---

4. Initialize DevStack:

```
$ ./stack.sh
```

   This will spew many screenfuls of text, and the first time you run it, it will download lots of software from the Internet. The output should eventually end with something like this:

```
This is your host IP address: 172.16.189.6
This is your host IPv6 address: ::1
Horizon is now available at http://172.16.189.6/dashboard
Keystone is serving at http://172.16.189.6/identity/
The default users are: admin and demo
The password: password
2017-03-09 15:10:54.117 | stack.sh completed in 2110 seconds.
```

   If there's some kind of failure, you can restart by running `./stack.sh` again. It won't restart exactly where it left off, but steps up to the one where it failed will skip the download steps. (Sometimes blindly restarting after a failure will allow it to succeed.) If you reboot your VM, you need to rerun this command. (If you run into trouble with `stack.sh` after rebooting your VM, try running `./unstack.sh`.)

   At this point you can navigate a web browser on your host to the Horizon dashboard URL. Many OpenStack operations can be initiated from this UI. Feel free to explore, but this tutorial focuses on the alternative command-line interfaces because they are easier to explain and to cut and paste.

5. The firewall in the VM by default allows SSH access but not HTTP. You will probably want HTTP access to use the OpenStack web interface. The following command enables that. (It also enables every other kind of network access, so if you're concerned about security then you might want to find a more targeted approach.)

```
$ sudo iptables -F
```

   (You need to re-run this if you reboot the VM.)

6. To use OpenStack command line utilities in the tutorial, run:

```
$ . ~/devstack/openrc admin
```

   This needs to be re-run each time you log in (but see the following section).

3.2.2 DevStack preliminaries

Before we really jump in, let's set up a couple of things in DevStack. This is the first real test that DevStack is working, so if you get errors from any of these commands, it's a sign that `stack.sh` didn't finish properly, or perhaps that you didn't run the `openrc admin` command at the end of the previous instructions.

If you stop and restart DevStack via `unstack.sh` followed by `stack.sh`, you have to rerun these steps.

1. For SSH access to the VMs we're going to create, we'll need a SSH keypair. Later on, we'll get OpenStack to install this keypair into VMs. Create one with:

```
$ openstack keypair create demo > ~/id_rsa_demo
$ chmod 600 ~/id_rsa_demo
```

2. By default, DevStack security groups drop incoming traffic, but to test networking in a reasonable way we need to enable it. You only need to actually edit one particular security group, but DevStack creates multiple and it's somewhat difficult to figure out which one is important because all of them are named "default". So, the following adds rules to allow SSH and ICMP traffic into **every** security group:

```
$ for group in $(openstack security group list -f value -c ID); do \
openstack security group rule create --ingress --ethertype IPv4 --dst-port 22 --
↪protocol tcp $group; \
openstack security group rule create --ingress --ethertype IPv4 --protocol ICMP
↪$group; \
done
```

3. Later on, we're going to create some VMs and we'll need an operating system image to install. DevStack comes with a very simple image built-in, called "cirros", which works fine. We need to get the UUID for this image. Our later commands assume shell variable `IMAGE_ID` holds this UUID. You can set this by hand, e.g.:

```
$ openstack image list
+--------------------------------------+--------------------------+--------+
| ID                                   | Name                     | Status |
+--------------------------------------+--------------------------+--------+
| 77f37d2c-3d6b-4e99-a01b-1fa5d78d1fa1 | cirros-0.3.5-x86_64-disk | active |
+--------------------------------------+--------------------------+--------+
$ IMAGE_ID=73ca34f3-63c4-4c10-a62f-4540afc24eaa
```

   or by parsing CLI output:

```
$ IMAGE_ID=$(openstack image list -f value -c ID)
```

Note: Your image ID will differ from the one above, as will every UUID in this tutorial. They will also change every time you run `stack.sh`. The UUIDs are generated randomly.

3.2.3 Shortening UUIDs

OpenStack, OVN, and Open vSwitch all really like UUIDs. These are great for uniqueness, but 36-character strings are terrible for readability. Statistically, just the first few characters are enough for uniqueness in small environments, so let's define a helper to make things more readable:

```
$ abbrev() { a='[0-9a-fA-F]' b=$a$a c=$b$b; sed "s/$b-$c-$c-$c$c$c//g"; }
```

You can use this as a filter to abbreviate UUIDs. For example, use it to abbreviate the above image list:

```
$ openstack image list -f yaml | abbrev
- ID: 77f37d
  Name: cirros-0.3.5-x86_64-disk
  Status: active
```

The command above also adds `-f yaml` to switch to YAML output format, because abbreviating UUIDs screws up the default table-based formatting and because YAML output doesn't produce wrap columns across lines and therefore is easier to cut and paste.

3.2.4 Overview

Now that DevStack is ready, with OVN set up as the networking back-end, here's an overview of what we're going to do in the remainder of the demo, all via OpenStack:

1. Switching: Create an OpenStack network `n1` and VMs `a` and `b` attached to it.

   An OpenStack network is a virtual switch; it corresponds to an OVN logical switch.

2. Routing: Create a second OpenStack network `n2` and VM `c` attached to it, then connect it to network `n1` by creating an OpenStack router and attaching `n1` and `n2` to it.
3. Gateways: Make VMs `a` and `b` available via an external network.
4. IPv6: Add IPv6 addresses to our VMs to demonstrate OVN support for IPv6 routing.
5. ACLs: Add and modify OpenStack stateless and stateful rules in security groups.
6. DHCP: How it works in OVN.
7. Further directions: Adding more compute nodes.

At each step, we will take a look at how the features in question work from OpenStack's Neutron networking layer at the top to the data plane layer at the bottom. From the highest to lowest level, these layers and the software components that connect them are:

- OpenStack Neutron, which as the top level in the system is the authoritative source of the virtual network configuration.

  We will use OpenStack's `openstack` utility to observe and modify Neutron and other OpenStack configuration.

- networking-ovn, the Neutron driver that interfaces with OVN and translates the internal Neutron representation of the virtual network into OVN's representation and pushes that representation down the OVN northbound database.

  In this tutorial it's rarely worth distinguishing Neutron from networking-ovn, so we usually don't break out this layer separately.

- The OVN Northbound database, aka NB DB. This is an instance of OVSDB, a simple general-purpose database that is used for multiple purposes in Open vSwitch and OVN. The NB DB's schema is in terms of networking concepts such as switches and routers. The NB DB serves the purpose that in other systems might be filled by some kind of API; for example, in place of calling an API to create or delete a logical switch, networking-ovn performs these operations by inserting or deleting a row in the NB DB's Logical_Switch table.

  We will use OVN's `ovn-nbctl` utility to observe the NB DB. (We won't directly modify data at this layer or below. Because configuration trickles down from Neutron through the stack, the right way to make changes is to use the `openstack` utility or another OpenStack interface and then wait for them to percolate through to lower layers.)

- The ovn-northd daemon, a program that runs centrally and translates the NB DB's network representation into the lower-level representation used by the OVN Southbound database in the next layer. The details of this daemon are usually not of interest, although without it OVN will not work, so this tutorial does not often mention it.

- The OVN Southbound database, aka SB DB, which is also an OVSDB database. Its schema is very different from the NB DB. Instead of familiar networking concepts, the SB DB defines the network in terms of collections of match-action rules called "logical flows", which while similar in concept to OpenFlow flows use logical concepts, such as virtual machine instances, in place of physical concepts like physical Ethernet ports.

  We will use OVN's `ovn-sbctl` utility to observe the SB DB.

- The ovn-controller daemon. A copy of ovn-controller runs on each hypervisor. It reads logical flows from the SB DB, translates them into OpenFlow flows, and sends them to Open vSwitch's ovs-vswitchd daemon. Like ovn-northd, usually the details of what this daemon are not of interest, even though it's important to the operation of the system.

- ovs-vswitchd. This program runs on each hypervisor. It is the core of Open vSwitch, which processes packets according to the OpenFlow flows set up by ovn-controller.

- Open vSwitch datapath. This is essentially a cache designed to accelerate packet processing. Open vSwitch includes a few different datapaths but OVN installations typically use one based on the Open vSwitch Linux kernel module.

3.2.5 Switching

Switching is the basis of networking in the real world and in virtual networking as well. OpenStack calls its concept of a virtual switch a "network", and OVN calls its corresponding concept a "logical switch".

In this step, we'll create an OpenStack network `n1`, then create VMs `a` and `b` and attach them to `n1`.

Creating network `n1`

Let's start by creating the network:

```
$ openstack network create --provider-network-type geneve n1
```

OpenStack needs to know the subnets that a network serves. We inform it by creating subnet objects. To keep it simple, let's give our network a single subnet for the 10.1.1.0/24 network. We have to give it a name, in this case `n1subnet`:

```
$ openstack subnet create --subnet-range 10.1.1.0/24 --network n1 n1subnet
```

If you ask Neutron to show us the available networks, we see `n1` as well as the two networks that DevStack creates by default:

```
$ openstack network list -f yaml | abbrev
- ID: 5b6baf
  Name: n1
  Subnets: 5e67e7
- ID: c02c4d
  Name: private
  Subnets: d88a34, fd87f9
- ID: d1ac28
  Name: public
  Subnets: 0b1e79, c87dc1
```

Neutron pushes this network setup down to the OVN northbound database. We can use `ovn-nbctl show` to see an overview of what's in the NB DB:

```
$ ovn-nbctl show | abbrev
switch 5b3d5f (neutron-c02c4d) (aka private)
    port b256dd
        type: router
        router-port: lrp-b256dd
    port f264e7
        type: router
        router-port: lrp-f264e7
switch 2579f4 (neutron-d1ac28) (aka public)
    port provnet-d1ac28
        type: localnet
        addresses: ["unknown"]
    port ae9b52
        type: router
        router-port: lrp-ae9b52
```

(continues on next page)

(continued from previous page)

```
switch 3eb263 (neutron-5b6baf) (aka n1)
router c59ad2 (neutron-9b057f) (aka router1)
    port lrp-ae9b52
        mac: "fa:16:3e:b2:d2:67"
        networks: ["172.24.4.9/24", "2001:db8::b/64"]
    port lrp-b256dd
        mac: "fa:16:3e:35:33:db"
        networks: ["fdb0:5860:4ba8::1/64"]
    port lrp-f264e7
        mac: "fa:16:3e:fc:c8:da"
        networks: ["10.0.0.1/26"]
    nat 80914c
        external ip: "172.24.4.9"
        logical ip: "10.0.0.0/26"
        type: "snat"
```

This output shows that OVN has three logical switches, each of which corresponds to a Neutron network, and a logical router that corresponds to the Neutron router that DevStack creates by default. The logical switch that corresponds to our new network `n1` has no ports yet, because we haven't added any. The `public` and `private` networks that DevStack creates by default have router ports that connect to the logical router.

Using ovn-northd, OVN translates the NB DB's high-level switch and router concepts into lower-level concepts of "logical datapaths" and logical flows. There's one logical datapath for each logical switch or router:

```
$ ovn-sbctl list datapath_binding | abbrev
_uuid               : 0ad69d
external_ids        : {logical-switch="5b3d5f", name="neutron-c02c4d", "name2
↪"=private}
tunnel_key          : 1

_uuid               : a8a758
external_ids        : {logical-switch="3eb263", name="neutron-5b6baf", "name2"="n1"}
tunnel_key          : 4

_uuid               : 191256
external_ids        : {logical-switch="2579f4", name="neutron-d1ac28", "name2"=public}
tunnel_key          : 3

_uuid               : b87bec
external_ids        : {logical-router="c59ad2", name="neutron-9b057f", "name2"=
↪"router1"}
tunnel_key          : 2
```

This output lists the NB DB UUIDs in external_ids:logical-switch and Neutron UUIDs in externals_ids:uuid. We can dive in deeper by viewing the OVN logical flows that implement a logical switch. Our new logical switch is a simple and almost pathological example given that it doesn't yet have any ports attached to it. We'll look at the details a bit later:

```
$ ovn-sbctl lflow-list n1 | abbrev
Datapath: "neutron-5b6baf" aka "n1" (a8a758)  Pipeline: ingress
  table=0 (ls_in_port_sec_l2  ), priority=100  , match=(eth.src[40]), action=(drop;)
  table=0 (ls_in_port_sec_l2  ), priority=100  , match=(vlan.present), action=(drop;)
...
Datapath: "neutron-5b6baf" aka "n1" (a8a758)  Pipeline: egress
  table=0 (ls_out_pre_lb      ), priority=0    , match=(1), action=(next;)
  table=1 (ls_out_pre_acl     ), priority=0    , match=(1), action=(next;)
...
```

We have one hypervisor (aka "compute node", in OpenStack parlance), which is the one where we're running all these commands. On this hypervisor, ovn-controller is translating OVN logical flows into OpenFlow flows ("physical flows"). It makes sense to go deeper, to see the OpenFlow flows that get generated from this datapath. By adding `--ovs` to the `ovn-sbctl` command, we can see OpenFlow flows listed just below their logical flows. We also need to use `sudo` because connecting to Open vSwitch is privileged. Go ahead and try it:

```
$ sudo ovn-sbctl --ovs lflow-list n1 | abbrev
Datapath: "neutron-5b6baf" aka "n1" (a8a758)  Pipeline: ingress
  table=0 (ls_in_port_sec_l2  ), priority=100  , match=(eth.src[40]), action=(drop;)
  table=0 (ls_in_port_sec_l2  ), priority=100  , match=(vlan.present), action=(drop;)
...
Datapath: "neutron-5b6baf" aka "n1" (a8a758)  Pipeline: egress
  table=0 (ls_out_pre_lb      ), priority=0    , match=(1), action=(next;)
  table=1 (ls_out_pre_acl     ), priority=0    , match=(1), action=(next;)
...
```

You were probably disappointed: the output didn't change, and no OpenFlow flows were printed. That's because no OpenFlow flows are installed for this logical datapath, which in turn is because there are no VIFs for this logical datapath on the local hypervisor. For a better example, you can try `ovn-sbctl --ovs` on one of the other logical datapaths.

Attaching VMs

A switch without any ports is not very interesting. Let's create a couple of VMs and attach them to the switch. Run the following commands, which create VMs named `a` and `b` and attaches them to our network `n1` with IP addresses 10.1.1.5 and 10.1.1.6, respectively. It is not actually necessary to manually assign IP address assignments, since OpenStack is perfectly happy to assign them itself from the subnet's IP address range, but predictable addresses are useful for our discussion:

```
$ openstack server create --nic net-id=n1,v4-fixed-ip=10.1.1.5 --flavor m1.nano --
↪image $IMAGE_ID --key-name demo a
$ openstack server create --nic net-id=n1,v4-fixed-ip=10.1.1.6 --flavor m1.nano --
↪image $IMAGE_ID --key-name demo b
```

These commands return before the VMs are really finished being built. You can run `openstack server list` a few times until each of them is shown in the state ACTIVE, which means that they're not just built but already running on the local hypervisor.

These operations had the side effect of creating separate "port" objects, but without giving those ports any easy-to-read names. It'll be easier to deal with them later if we can refer to them by name, so let's name `a`'s port `ap` and `b`'s port `bp`:

```
$ openstack port set --name ap $(openstack port list --server a -f value -c ID)
$ openstack port set --name bp $(openstack port list --server b -f value -c ID)
```

We'll need to refer to these ports' MAC addresses a few times, so let's put them in variables:

```
$ AP_MAC=$(openstack port show -f value -c mac_address ap)
$ BP_MAC=$(openstack port show -f value -c mac_address bp)
```

At this point you can log into the consoles of the VMs if you like. You can do that from the OpenStack web interface or get a direct URL to paste into a web browser using a command like:

```
$ openstack console url show -f yaml a
```

(The option `-f yaml` keeps the URL in the output from being broken into noncontiguous pieces on a 80-column console.)

The VMs don't have many tools in them but `ping` and `ssh` from one to the other should work fine. The VMs do not have any external network access or DNS configuration.

Let's chase down what's changed in OVN. Start with the NB DB at the top of the system. It's clear that our logical switch now has the two logical ports attached to it:

```
$ ovn-nbctl show | abbrev
...
switch 3eb263 (neutron-5b6baf) (aka n1)
    port c29d41 (aka bp)
        addresses: ["fa:16:3e:99:7a:17 10.1.1.6"]
    port 820c08 (aka ap)
        addresses: ["fa:16:3e:a9:4c:c7 10.1.1.5"]
...
```

We can get some more details on each of these by looking at their NB DB records in the Logical_Switch_Port table. Each port has addressing information, port security enabled, and a pointer to DHCP configuration (which we'll look at much later in *DHCP*):

```
$ ovn-nbctl list logical_switch_port ap bp | abbrev
_uuid               : ef17e5
addresses           : ["fa:16:3e:a9:4c:c7 10.1.1.5"]
dhcpv4_options      : 165974
dhcpv6_options      : []
dynamic_addresses   : []
enabled             : true
external_ids        : {"neutron:port_name"=ap}
name                : "820c08"
options             : {}
parent_name         : []
port_security       : ["fa:16:3e:a9:4c:c7 10.1.1.5"]
tag                 : []
tag_request         : []
type                : ""
up                  : true

_uuid               : e8af12
addresses           : ["fa:16:3e:99:7a:17 10.1.1.6"]
dhcpv4_options      : 165974
dhcpv6_options      : []
dynamic_addresses   : []
enabled             : true
external_ids        : {"neutron:port_name"=bp}
name                : "c29d41"
options             : {}
parent_name         : []
port_security       : ["fa:16:3e:99:7a:17 10.1.1.6"]
tag                 : []
tag_request         : []
type                : ""
up                  : true
```

Now that the logical switch is less pathological, it's worth taking another look at the SB DB logical flow table. Try a command like this:

```
$ ovn-sbctl lflow-list n1 | abbrev | less -S
```

and then glance through the flows. Packets that egress a VM into the logical switch travel through the flow table's ingress pipeline starting from table 0. At each table, the switch finds the highest-priority logical flow that matches and executes its actions, or if there's no matching flow then the packet is dropped. The ovn-sb(5) manpage gives all the details, but with a little thought it's possible to guess a lot without reading the manpage. For example, consider the flows in ingress pipeline table 0, which are the first flows encountered by a packet traversing the switch:

```
table=0 (ls_in_port_sec_l2  ), priority=100  , match=(eth.src[40]), action=(drop;)
table=0 (ls_in_port_sec_l2  ), priority=100  , match=(vlan.present), action=(drop;)
table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport == "820c08" && eth.src␣
↪== {fa:16:3e:a9:4c:c7}), action=(next;)
table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport == "c29d41" && eth.src␣
↪== {fa:16:3e:99:7a:17}), action=(next;)
```

The first two flows, with priority 100, immediately drop two kinds of invalid packets: those with a multicast or broadcast Ethernet source address (since multicast is only for packet destinations) and those with a VLAN tag (because OVN doesn't yet support VLAN tags inside logical networks). The next two flows implement L2 port security: they advance to the next table for packets with the correct Ethernet source addresses for their ingress ports. A packet that does not match any flow is implicitly dropped, so there's no need for flows to deal with mismatches.

The logical flow table includes many other flows, some of which we will look at later. For now, it's most worth looking at ingress table 13:

```
table=13(ls_in_l2_lkup      ), priority=100  , match=(eth.mcast), action=(outport = "_
↪MC_flood"; output;)
table=13(ls_in_l2_lkup      ), priority=50   , match=(eth.dst == fa:16:3e:99:7a:17),␣
↪action=(outport = "c29d41"; output;)
table=13(ls_in_l2_lkup      ), priority=50   , match=(eth.dst == fa:16:3e:a9:4c:c7),␣
↪action=(outport = "820c08"; output;)
```

The first flow in table 13 checks whether the packet is an Ethernet multicast or broadcast and, if so, outputs it to a special port that egresses to every logical port (other than the ingress port). Otherwise the packet is output to the port corresponding to its Ethernet destination address. Packets addressed to any other Ethernet destination are implicitly dropped.

(It's common for an OVN logical switch to know all the MAC addresses supported by its logical ports, like this one. That's why there's no logic here for MAC learning or flooding packets to unknown MAC addresses. OVN does support unknown MAC handling but that's not in play in our example.)

Note: If you're interested in the details for the multicast group, you can run a command like the following and then look at the row for the correct datapath:

```
$ ovn-sbctl find multicast_group name=_MC_flood | abbrev
```

Now if you want to look at the OpenFlow flows, you can actually see them. For example, here's the beginning of the output that lists the first four logical flows, which we already looked at above, and their corresponding OpenFlow flows. If you want to know more about the syntax, the ovs-fields(7) manpage explains OpenFlow matches and ovs-ofctl(8) explains OpenFlow actions:

```
$ sudo ovn-sbctl --ovs lflow-list n1 | abbrev
Datapath: "neutron-5b6baf" aka "n1" (a8a758)  Pipeline: ingress
  table=0 (ls_in_port_sec_l2  ), priority=100  , match=(eth.src[40]), action=(drop;)
    table=8 metadata=0x4,dl_src=01:00:00:00:00:00/01:00:00:00:00:00 actions=drop
```

(continues on next page)

(continued from previous page)

```
  table=0 (ls_in_port_sec_l2  ), priority=100  , match=(vlan.present), action=(drop;)
    table=8 metadata=0x4,vlan_tci=0x1000/0x1000 actions=drop
  table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport == "820c08" && eth.src
↪== {fa:16:3e:a9:4c:c7}), action=(next;)
    table=8 reg14=0x1,metadata=0x4,dl_src=fa:16:3e:a9:4c:c7 actions=resubmit(,9)
  table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport == "c29d41" && eth.src
↪== {fa:16:3e:99:7a:17}), action=(next;)
    table=8 reg14=0x2,metadata=0x4,dl_src=fa:16:3e:99:7a:17 actions=resubmit(,9)
...
```

Logical Tracing

Let's go a level deeper. So far, everything we've done has been fairly general. We can also look at something more specific: the path that a particular packet would take through OVN, logically, and Open vSwitch, physically.

Let's use OVN's ovn-trace utility to see what happens to packets from a logical point of view. The `ovn-trace`(8) manpage has a lot of detail on how to do that, but let's just start by building up from a simple example. You can start with a command that just specifies the logical datapath, an input port, and nothing else; unspecified fields default to all-zeros. This doesn't do much:

```
$ ovn-trace n1 'inport == "ap"'
...
ingress(dp="n1", inport="ap")
-----------------------------
 0. ls_in_port_sec_l2: no match (implicit drop)
```

We see that the packet was dropped in logical table 0, "ls_in_port_sec_l2", the L2 port security stage (as we discussed earlier). That's because we didn't use the right Ethernet source address for `a`. Let's see what happens if we do:

```
$ ovn-trace n1 'inport == "ap" && eth.src == '$AP_MAC
...
ingress(dp="n1", inport="ap")
-----------------------------
 0. ls_in_port_sec_l2 (northd.c:3234): inport == "ap" && eth.src ==
↪{fa:16:3e:a9:4c:c7}, priority 50, uuid 6dcc418a
    next;
13. ls_in_l2_lkup: no match (implicit drop)
```

Now the packet passes through L2 port security and skips through several other tables until it gets dropped in the L2 lookup stage (because the destination is unknown). Let's add the Ethernet destination for `b`:

```
$ ovn-trace n1 'inport == "ap" && eth.src == '$AP_MAC' && eth.dst == '$BP_MAC
...
ingress(dp="n1", inport="ap")
-----------------------------
 0. ls_in_port_sec_l2 (northd.c:3234): inport == "ap" && eth.src ==
↪{fa:16:3e:a9:4c:c7}, priority 50, uuid 6dcc418a
    next;
13. ls_in_l2_lkup (northd.c:3529): eth.dst == fa:16:3e:99:7a:17, priority 50, uuid
↪57a4c46f
    outport = "bp";
    output;

egress(dp="n1", inport="ap", outport="bp")
```

(continues on next page)

(continued from previous page)

```
----------------------------------------
 8. ls_out_port_sec_l2 (northd.c:3654): outport == "bp" && eth.dst ==
↪{fa:16:3e:99:7a:17}, priority 50, uuid 8aa6426d
    output;
    /* output to "bp", type "" */
```

You can see that in this case the packet gets properly switched from `a` to `b`.

Physical Tracing for Hypothetical Packets

ovn-trace showed us how a hypothetical packet would travel through the system in a logical fashion, that is, without regard to how VMs are distributed across the physical network. This is a convenient representation for understanding how OVN is **supposed** to work abstractly, but sometimes we might want to know more about how it actually works in the real systems where it is running. For this, we can use the tracing tool that Open vSwitch provides, which traces a hypothetical packet through the OpenFlow tables.

We can actually get two levels of detail. Let's start with the version that's easier to interpret, by physically tracing a packet that looks like the one we logically traced before. One obstacle is that we need to know the OpenFlow port number of the input port. One way to do that is to look for a port whose "attached-mac" is the one we expect and print its ofport number:

```
$ AP_PORT=$(ovs-vsctl --bare --columns=ofport find  interface external-ids:attached-
↪mac=\"$AP_MAC\")
$ echo $AP_PORT
3
```

(You could also just do a plain `ovs-vsctl list interface` and then look through for the right row and pick its `ofport` value.)

Now we can feed this input port number into `ovs-appctl ofproto/trace` along with the correct Ethernet source and destination addresses and get a physical trace:

```
$ sudo ovs-appctl ofproto/trace br-int in_port=$AP_PORT,dl_src=$AP_MAC,dl_dst=$BP_MAC
Flow: in_port=3,vlan_tci=0x0000,dl_src=fa:16:3e:a9:4c:c7,dl_dst=fa:16:3e:99:7a:17,dl_
↪type=0x0000

bridge("br-int")
----------------
 0. in_port=3, priority 100
    set_field:0x8->reg13
    set_field:0x9->reg11
    set_field:0xa->reg12
    set_field:0x4->metadata
    set_field:0x1->reg14
    resubmit(,8)
 8. reg14=0x1,metadata=0x4,dl_src=fa:16:3e:a9:4c:c7, priority 50, cookie 0x6dcc418a
    resubmit(,9)
 9. metadata=0x4, priority 0, cookie 0x8fe8689e
    resubmit(,10)
10. metadata=0x4, priority 0, cookie 0x719549d1
    resubmit(,11)
11. metadata=0x4, priority 0, cookie 0x39c99e6f
    resubmit(,12)
12. metadata=0x4, priority 0, cookie 0x838152a3
    resubmit(,13)
```

(continues on next page)

(continued from previous page)

```
13. metadata=0x4, priority 0, cookie 0x918259e3
    resubmit(,14)
14. metadata=0x4, priority 0, cookie 0xcad14db2
    resubmit(,15)
15. metadata=0x4, priority 0, cookie 0x7834d912
    resubmit(,16)
16. metadata=0x4, priority 0, cookie 0x87745210
    resubmit(,17)
17. metadata=0x4, priority 0, cookie 0x34951929
    resubmit(,18)
18. metadata=0x4, priority 0, cookie 0xd7a8c9fb
    resubmit(,19)
19. metadata=0x4, priority 0, cookie 0xd02e9578
    resubmit(,20)
20. metadata=0x4, priority 0, cookie 0x42d35507
    resubmit(,21)
21. metadata=0x4,dl_dst=fa:16:3e:99:7a:17, priority 50, cookie 0x57a4c46f
    set_field:0x2->reg15
    resubmit(,32)
32. priority 0
    resubmit(,33)
33. reg15=0x2,metadata=0x4, priority 100
    set_field:0xb->reg13
    set_field:0x9->reg11
    set_field:0xa->reg12
    resubmit(,34)
34. priority 0
    set_field:0->reg0
    set_field:0->reg1
    set_field:0->reg2
    set_field:0->reg3
    set_field:0->reg4
    set_field:0->reg5
    set_field:0->reg6
    set_field:0->reg7
    set_field:0->reg8
    set_field:0->reg9
    resubmit(,40)
40. metadata=0x4, priority 0, cookie 0xde9f3899
    resubmit(,41)
41. metadata=0x4, priority 0, cookie 0x74074eff
    resubmit(,42)
42. metadata=0x4, priority 0, cookie 0x7789c8b1
    resubmit(,43)
43. metadata=0x4, priority 0, cookie 0xa6b002c0
    resubmit(,44)
44. metadata=0x4, priority 0, cookie 0xaeab2b45
    resubmit(,45)
45. metadata=0x4, priority 0, cookie 0x290cc4d4
    resubmit(,46)
46. metadata=0x4, priority 0, cookie 0xa3223b88
    resubmit(,47)
47. metadata=0x4, priority 0, cookie 0x7ac2132e
    resubmit(,48)
48. reg15=0x2,metadata=0x4,dl_dst=fa:16:3e:99:7a:17, priority 50, cookie 0x8aa6426d
    resubmit(,64)
64. priority 0
```

(continues on next page)

(continued from previous page)

```
    resubmit(,65)
65. reg15=0x2,metadata=0x4, priority 100
    output:4

Final flow: reg11=0x9,reg12=0xa,reg13=0xb,reg14=0x1,reg15=0x2,metadata=0x4,in_port=3,
↪vlan_tci=0x0000,dl_src=fa:16:3e:a9:4c:c7,dl_dst=fa:16:3e:99:7a:17,dl_type=0x0000
Megaflow: recirc_id=0,ct_state=-new-est-rel-rpl-inv-trk,ct_label=0/0x1,in_port=3,vlan_
↪tci=0x0000/0x1000,dl_src=fa:16:3e:a9:4c:c7,dl_dst=fa:16:3e:99:7a:17,dl_type=0x0000
Datapath actions: 4
```

There's a lot there, which you can read through if you like, but the important part is:

```
65. reg15=0x2,metadata=0x4, priority 100
    output:4
```

which means that the packet is ultimately being output to OpenFlow port 4. That's port `b`, which you can confirm with:

```
$ sudo ovs-vsctl find interface ofport=4
_uuid               : 840a5aca-ea8d-4c16-a11b-a94e0f408091
admin_state         : up
bfd                 : {}
bfd_status          : {}
cfm_fault           : []
cfm_fault_status    : []
cfm_flap_count      : []
cfm_health          : []
cfm_mpid            : []
cfm_remote_mpids    : []
cfm_remote_opstate  : []
duplex              : full
error               : []
external_ids        : {attached-mac="fa:16:3e:99:7a:17", iface-id="c29d4120-20a4-4c44-
↪bd83-8d91f5f447fd", iface-status=active, vm-id="2db969ca-ca2a-4d9a-b49e-f287d39c5645
↪"}
ifindex             : 9
ingress_policing_burst: 0
ingress_policing_rate: 0
lacp_current        : []
link_resets         : 1
link_speed          : 10000000
link_state          : up
lldp                : {}
mac                 : []
mac_in_use          : "fe:16:3e:99:7a:17"
mtu                 : 1500
mtu_request         : []
name                : "tapc29d4120-20"
ofport              : 4
ofport_request      : []
options             : {}
other_config        : {}
statistics          : {collisions=0, rx_bytes=4254, rx_crc_err=0, rx_dropped=0, rx_
↪errors=0, rx_frame_err=0, rx_over_err=0, rx_packets=39, tx_bytes=4188, tx_dropped=0,
↪ tx_errors=0, tx_packets=39}
status              : {driver_name=tun, driver_version="1.6", firmware_version=""}
type                : ""
```

or:

```
$ BP_PORT=$(ovs-vsctl --bare --columns=ofport find  interface external-ids:attached-
↪mac=\"$BP_MAC\")
$ echo $BP_PORT
4
```

Physical Tracing for Real Packets

In the previous sections we traced a hypothetical L2 packet, one that's honestly not very realistic: we didn't even supply an Ethernet type, so it defaulted to zero, which isn't anything one would see on a real network. We could refine our packet so that it becomes a more realistic TCP or UDP or ICMP, etc. packet, but let's try a different approach: working from a real packet.

Pull up a console for VM `a` and start `ping 10.1.1.6`, then leave it running for the rest of our experiment.

Now go back to your DevStack session and run:

```
$ sudo watch ovs-dpctl dump-flows
```

We're working with a new program. ovn-dpctl is an interface to Open vSwitch datapaths, in this case to the Linux kernel datapath. Its `dump-flows` command displays the contents of the in-kernel flow cache, and by running it under the `watch` program we see a new snapshot of the flow table every 2 seconds.

Look through the output for a flow that begins with `recirc_id(0)` and matches the Ethernet source address for `a`. There is one flow per line, but the lines are very long, so it's easier to read if you make the window very wide. This flow's packet counter should be increasing at a rate of 1 packet per second. It looks something like this:

```
recirc_id(0),in_port(3),eth(src=fa:16:3e:f5:2a:90),eth_type(0x0800),ipv4(src=10.1.1.5,
↪frag=no), packets:388, bytes:38024, used:0.977s, actions:ct(zone=8),recirc(0x18)
```

Note: Flows in the datapath can expire quickly and the `watch` command mentioned above may be too slow to catch it. If that is your case, stop the `ping 10.1.1.6` session and re-start it a few seconds after this command:

```
$ sudo conntrack -F ; rm -f /tmp/flows.txt ; \
     for _ in $(seq 100) ; do \
     sudo ovs-dpctl dump-flows >> /tmp/flows.txt ; \
     sleep 0.1 ; done
```

Then, look for `recirc_id(0)` in flows.txt after ping command was issued:

```
$ sort --uniq /tmp/flows.txt | grep zone
```

We can hand the first part of this (everything up to the first space) to `ofproto/trace`, and it will tell us what happens:

```
$ sudo ovs-appctl ofproto/trace 'recirc_id(0),in_port(3),eth(src=fa:16:3e:a9:4c:c7),
↪eth_type(0x0800),ipv4(src=10.1.1.5,dst=10.1.0.0/255.255.0.0,frag=no)'
Flow: ip,in_port=3,vlan_tci=0x0000,dl_src=fa:16:3e:a9:4c:c7,dl_dst=00:00:00:00:00:00,
↪nw_src=10.1.1.5,nw_dst=10.1.0.0,nw_proto=0,nw_tos=0,nw_ecn=0,nw_ttl=0

bridge("br-int")
----------------
 0. in_port=3, priority 100
```

(continues on next page)

(continued from previous page)

```
    set_field:0x8->reg13
    set_field:0x9->reg11
    set_field:0xa->reg12
    set_field:0x4->metadata
    set_field:0x1->reg14
    resubmit(,8)
 8. reg14=0x1,metadata=0x4,dl_src=fa:16:3e:a9:4c:c7, priority 50, cookie 0x6dcc418a
    resubmit(,9)
 9. ip,reg14=0x1,metadata=0x4,dl_src=fa:16:3e:a9:4c:c7,nw_src=10.1.1.5, priority 90,␣
↪cookie 0x343af48c
    resubmit(,10)
10. metadata=0x4, priority 0, cookie 0x719549d1
    resubmit(,11)
11. ip,metadata=0x4, priority 100, cookie 0x46c089e6
    load:0x1->NXM_NX_XXREG0[96]
    resubmit(,12)
12. metadata=0x4, priority 0, cookie 0x838152a3
    resubmit(,13)
13. ip,reg0=0x1/0x1,metadata=0x4, priority 100, cookie 0xd1941634
    ct(table=22,zone=NXM_NX_REG13[0..15])
    drop

Final flow: ip,reg0=0x1,reg11=0x9,reg12=0xa,reg13=0x8,reg14=0x1,metadata=0x4,in_
↪port=3,vlan_tci=0x0000,dl_src=fa:16:3e:a9:4c:c7,dl_dst=00:00:00:00:00:00,nw_src=10.
↪1.1.5,nw_dst=10.1.0.0,nw_proto=0,nw_tos=0,nw_ecn=0,nw_ttl=0
Megaflow: recirc_id=0,ip,in_port=3,vlan_tci=0x0000/0x1000,dl_src=fa:16:3e:a9:4c:c7,nw_
↪src=10.1.1.5,nw_dst=10.1.0.0/16,nw_frag=no
Datapath actions: ct(zone=8),recirc(0xb)
```

Note: Be careful cutting and pasting `ovs-dpctl dump-flows` output into `ofproto/trace` because the latter has terrible error reporting. If you add an extra line break, etc., it will likely give you a useless error message.

There's no `output` action in the output, but there are `ct` and `recirc` actions (which you can see in the `Datapath actions` at the end). The `ct` action tells the kernel to pass the packet through the kernel connection tracking for firewalling purposes and the `recirc` says to go back to the flow cache for another pass based on the firewall results. The `0xb` value inside the `recirc` gives us a hint to look at the kernel flows for a cached flow with `recirc_id(0xb)`. Indeed, there is one:

```
recirc_id(0xb),in_port(3),ct_state(-new+est-rel-rpl-inv+trk),ct_label(0/0x1),
↪eth(src=fa:16:3e:a9:4c:c7,dst=fa:16:3e:99:7a:17),eth_type(0x0800),ipv4(dst=10.1.1.4/
↪255.255.255.252,frag=no), packets:171, bytes:16758, used:0.271s,␣
↪actions:ct(zone=11),recirc(0xc)
```

We can then repeat our command with the match part of this kernel flow:

```
$ sudo ovs-appctl ofproto/trace 'recirc_id(0xb),in_port(3),ct_state(-new+est-rel-rpl-
↪inv+trk),ct_label(0/0x1),eth(src=fa:16:3e:a9:4c:c7,dst=fa:16:3e:99:7a:17),eth_
↪type(0x0800),ipv4(dst=10.1.1.4/255.255.255.252,frag=no)'
...
Datapath actions: ct(zone=11),recirc(0xc)
```

In other words, the flow passes through the connection tracker a second time. The first time was for `a`'s outgoing firewall; this second time is for `b`'s incoming firewall. Again, we continue tracing with `recirc_id(0xc)`:

```
$ sudo ovs-appctl ofproto/trace 'recirc_id(0xc),in_port(3),ct_state(-new+est-rel-rpl-
↪inv+trk),ct_label(0/0x1),eth(src=fa:16:3e:a9:4c:c7,dst=fa:16:3e:99:7a:17),eth_
↪type(0x0800),ipv4(dst=10.1.1.6,proto=1,frag=no)'
...
Datapath actions: 4
```

It took multiple hops, but we finally came to the end of the line where the packet was output to `b` after passing through both firewalls. The port number here is a datapath port number, which is usually different from an OpenFlow port number. To check that it is `b`'s port, we first list the datapath ports to get the name corresponding to the port number:

```
$ sudo ovs-dpctl show
system@ovs-system:
        lookups: hit:1994 missed:56 lost:0
        flows: 6
        masks: hit:2340 total:4 hit/pkt:1.14
        port 0: ovs-system (internal)
        port 1: br-int (internal)
        port 2: br-ex (internal)
        port 3: tap820c0888-13
        port 4: tapc29d4120-20
```

and then confirm that this is the port we think it is with a command like this:

```
$ ovs-vsctl --columns=external-ids list interface tapc29d4120-20
external_ids        : {attached-mac="fa:16:3e:99:7a:17", iface-id="c29d4120-20a4-4c44-
↪bd83-8d91f5f447fd", iface-status=active, vm-id="2db969ca-ca2a-4d9a-b49e-f287d39c5645
↪"}
```

Finally, we can relate the OpenFlow flows from our traces back to OVN logical flows. For individual flows, cut and paste a "cookie" value from `ofproto/trace` output into `ovn-sbctl lflow-list`, e.g.:

```
$ ovn-sbctl lflow-list 0x6dcc418a|abbrev
Datapath: "neutron-5b6baf" aka "n1" (a8a758)  Pipeline: ingress
  table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport == "820c08" && eth.src␣
↪== {fa:16:3e:a9:4c:c7}), action=(next;)
```

Or, you can pipe `ofproto/trace` output through `ovn-detrace` to annotate every flow:

```
$ sudo ovs-appctl ofproto/trace 'recirc_id(0xc),in_port(3),ct_state(-new+est-rel-rpl-
↪inv+trk),ct_label(0/0x1),eth(src=fa:16:3e:a9:4c:c7,dst=fa:16:3e:99:7a:17),eth_
↪type(0x0800),ipv4(dst=10.1.1.6,proto=1,frag=no)' | ovn-detrace
...
```

3.2.6 Routing

Previously we set up a pair of VMs `a` and `b` on a network `n1` and demonstrated how packets make their way between them. In this step, we'll set up a second network `n2` with a new VM `c`, connect a router `r` to both networks, and demonstrate how routing works in OVN.

There's nothing really new for the network and the VM so let's just go ahead and create them:

```
$ openstack network create --provider-network-type geneve n2
$ openstack subnet create --subnet-range 10.1.2.0/24 --network n2 n2subnet
$ openstack server create --nic net-id=n2,v4-fixed-ip=10.1.2.7 --flavor m1.nano --
↪image $IMAGE_ID --key-name demo c
```

(continues on next page)

(continued from previous page)

```
$ openstack port set --name cp $(openstack port list --server c -f value -c ID)
$ CP_MAC=$(openstack port show -f value -c mac_address cp)
```

The new network `n2` is not yet connected to `n1` in any way. You can try tracing a packet from `a` to see, for example, that it doesn't make it to `c`. Instead, it will end up as `multicast unknown` in `n1`:

```
$ ovn-trace n1 'inport == "ap" && eth.src == '$AP_MAC' && eth.dst == '$CP_MAC
...
```

Now create an OpenStack router and connect it to `n1` and `n2`:

```
$ openstack router create r
$ openstack router add subnet r n1subnet
$ openstack router add subnet r n2subnet
```

Now `a`, `b`, and `c` should all be able to reach other. You can get some verification that routing is taking place by running you `ping` between `c` and one of the other VMs: the reported TTL should be one less than between `a` and `b` (63 instead of 64).

Observe via `ovn-nbctl` the new OVN logical switch and router and then ports that connect them together:

```
$ ovn-nbctl show|abbrev
...
switch f51234 (neutron-332346) (aka n2)
    port 82b983
        type: router
        router-port: lrp-82b983
    port 2e585f (aka cp)
        addresses: ["fa:16:3e:89:f2:36 10.1.2.7"]
switch 3eb263 (neutron-5b6baf) (aka n1)
    port c29d41 (aka bp)
        addresses: ["fa:16:3e:99:7a:17 10.1.1.6"]
    port 820c08 (aka ap)
        addresses: ["fa:16:3e:a9:4c:c7 10.1.1.5"]
    port 17d870
        type: router
        router-port: lrp-17d870
...
router dde06c (neutron-f88ebc) (aka r)
    port lrp-82b983
        mac: "fa:16:3e:19:9f:46"
        networks: ["10.1.2.1/24"]
    port lrp-17d870
        mac: "fa:16:3e:f6:e2:8f"
        networks: ["10.1.1.1/24"]
```

We have not yet looked at the logical flows for an OVN logical router. You might find it of interest to look at them on your own:

```
$ ovn-sbctl lflow-list r | abbrev | less -S
...
```

Let's grab the `n1subnet` router porter MAC address to simplify later commands:

```
$ N1SUBNET_MAC=$(ovn-nbctl --bare --columns=mac find logical_router_port networks=10.
↪1.1.1/24)
```

Let's see what happens at the logical flow level for an ICMP packet from `a` to `c`. This generates a long trace but an interesting one, so we'll look at it bit by bit. The first three stanzas in the output show the packet's ingress into `n1` and processing through the firewall on that side (via the "ct_next" connection-tracking action), and then the selection of the port that leads to router `r` as the output port:

```
$ ovn-trace n1 'inport == "ap" && eth.src == '$AP_MAC' && eth.dst == '$N1SUBNET_MAC' &
↪& ip4.src == 10.1.1.5 && ip4.dst == 10.1.2.7 && ip.ttl == 64 && icmp4.type == 8'
...
ingress(dp="n1", inport="ap")
-----------------------------
 0. ls_in_port_sec_l2 (northd.c:3234): inport == "ap" && eth.src ==
↪{fa:16:3e:a9:4c:c7}, priority 50, uuid 6dcc418a
    next;
 1. ls_in_port_sec_ip (northd.c:2364): inport == "ap" && eth.src == fa:16:3e:a9:4c:c7␣
↪&& ip4.src == {10.1.1.5}, priority 90, uuid 343af48c
    next;
 3. ls_in_pre_acl (northd.c:2646): ip, priority 100, uuid 46c089e6
    reg0[0] = 1;
    next;
 5. ls_in_pre_stateful (northd.c:2764): reg0[0] == 1, priority 100, uuid d1941634
    ct_next;

ct_next(ct_state=est|trk /* default (use --ct to customize) */)
---------------------------------------------------------------
 6. ls_in_acl (northd.c:2925): !ct.new && ct.est && !ct.rpl && ct_label.blocked == 0 &
↪& (inport == "ap" && ip4), priority 2002, uuid a12b39f0
    next;
13. ls_in_l2_lkup (northd.c:3529): eth.dst == fa:16:3e:f6:e2:8f, priority 50, uuid␣
↪c43ead31
    outport = "17d870";
    output;

egress(dp="n1", inport="ap", outport="17d870")
----------------------------------------------
 1. ls_out_pre_acl (northd.c:2626): ip && outport == "17d870", priority 110, uuid␣
↪60395450
    next;
 8. ls_out_port_sec_l2 (northd.c:3654): outport == "17d870", priority 50, uuid␣
↪91b5cab0
    output;
    /* output to "17d870", type "patch" */
```

The next two stanzas represent processing through logical router `r`. The processing in table 5 is the core of the routing implementation: it recognizes that the packet is destined for an attached subnet, decrements the TTL and updates the Ethernet source address. Table 6 then selects the Ethernet destination address based on the IP destination. The packet then passes to switch `n2` via an OVN "logical patch port":

```
ingress(dp="r", inport="lrp-17d870")
------------------------------------
 0. lr_in_admission (northd.c:4071): eth.dst == fa:16:3e:f6:e2:8f && inport == "lrp-
↪17d870", priority 50, uuid fa5270b0
    next;
 5. lr_in_ip_routing (northd.c:3782): ip4.dst == 10.1.2.0/24, priority 49, uuid␣
↪5f9d469f
    ip.ttl--;
    reg0 = ip4.dst;
    reg1 = 10.1.2.1;
```

(continues on next page)

(continued from previous page)

```
    eth.src = fa:16:3e:19:9f:46;
    outport = "lrp-82b983";
    flags.loopback = 1;
    next;
 6. lr_in_arp_resolve (northd.c:5088): outport == "lrp-82b983" && reg0 == 10.1.2.7,␣
↪priority 100, uuid 03d506d3
    eth.dst = fa:16:3e:89:f2:36;
    next;
 8. lr_in_arp_request (northd.c:5260): 1, priority 0, uuid 6dacdd82
    output;

egress(dp="r", inport="lrp-17d870", outport="lrp-82b983")
---------------------------------------------------------
 3. lr_out_delivery (northd.c:5288): outport == "lrp-82b983", priority 100, uuid␣
↪00bea4f2
    output;
    /* output to "lrp-82b983", type "patch" */
```

Finally the logical switch for `n2` runs through the same logic as `n1` and the packet is delivered to VM `c`:

```
ingress(dp="n2", inport="82b983")
---------------------------------
 0. ls_in_port_sec_l2 (northd.c:3234): inport == "82b983", priority 50, uuid 9a789e06
    next;
 3. ls_in_pre_acl (northd.c:2624): ip && inport == "82b983", priority 110, uuid␣
↪ab52f21a
    next;
13. ls_in_l2_lkup (northd.c:3529): eth.dst == fa:16:3e:89:f2:36, priority 50, uuid␣
↪dcafb3e9
    outport = "cp";
    output;

egress(dp="n2", inport="82b983", outport="cp")
----------------------------------------------
 1. ls_out_pre_acl (northd.c:2648): ip, priority 100, uuid cd9cfa74
    reg0[0] = 1;
    next;
 2. ls_out_pre_stateful (northd.c:2766): reg0[0] == 1, priority 100, uuid 9e8e22c5
    ct_next;

ct_next(ct_state=est|trk /* default (use --ct to customize) */)
---------------------------------------------------------------
 4. ls_out_acl (northd.c:2925): !ct.new && ct.est && !ct.rpl && ct_label.blocked == 0␣
↪&& (outport == "cp" && ip4 && ip4.src == $as_ip4_0fc1b6cf_f925_49e6_8f00_
↪6dd13beca9dc), priority 2002, uuid a746fa0d
    next;
 7. ls_out_port_sec_ip (northd.c:2364): outport == "cp" && eth.dst ==␣
↪fa:16:3e:89:f2:36 && ip4.dst == {255.255.255.255, 224.0.0.0/4, 10.1.2.7}, priority␣
↪90, uuid 4d9862b5
    next;
 8. ls_out_port_sec_l2 (northd.c:3654): outport == "cp" && eth.dst ==
↪{fa:16:3e:89:f2:36}, priority 50, uuid 0242cdc3
    output;
    /* output to "cp", type "" */
```

Physical Tracing

It's possible to use `ofproto/trace`, just as before, to trace a packet through OpenFlow tables, either for a hypothetical packet or one that you get from a real test case using `ovs-dpctl`. The process is just the same as before and the output is almost the same, too. Using a router doesn't actually introduce any interesting new wrinkles, so we'll skip over this for this case and for the remainder of the tutorial, but you can follow the steps on your own if you like.

3.2.7 Adding a Gateway

The VMs that we've created can access each other but they are isolated from the physical world. In OpenStack, the dominant way to connect a VM to external networks is by creating what is called a "floating IP address", which uses network address translation to connect an external address to an internal one.

DevStack created a pair of networks named "private" and "public". To use a floating IP address from a VM, we first add a port to the VM with an IP address from the "private" network, then we create a floating IP address on the "public" network, then we associate the port with the floating IP address.

Let's add a new VM `d` with a floating IP:

```
$ openstack server create --nic net-id=private --flavor m1.nano --image $IMAGE_ID --
↪key-name demo d
$ openstack port set --name dp $(openstack port list --server d -f value -c ID)
$ DP_MAC=$(openstack port show -f value -c mac_address dp)
$ openstack floating ip create --floating-ip-address 172.24.4.8 public
$ openstack server add floating ip d 172.24.4.8
```

(We specified a particular floating IP address to make the examples easier to follow, but without that OpenStack will automatically allocate one.)

It's also necessary to configure the "public" network because DevStack does not do it automatically:

```
$ sudo ip link set br-ex up
$ sudo ip addr add 172.24.4.1/24 dev br-ex
```

Now you should be able to "ping" VM `d` from the OpenStack host:

```
$ ping 172.24.4.8
PING 172.24.4.8 (172.24.4.8) 56(84) bytes of data.
64 bytes from 172.24.4.8: icmp_seq=1 ttl=63 time=56.0 ms
64 bytes from 172.24.4.8: icmp_seq=2 ttl=63 time=1.44 ms
64 bytes from 172.24.4.8: icmp_seq=3 ttl=63 time=1.04 ms
64 bytes from 172.24.4.8: icmp_seq=4 ttl=63 time=0.403 ms
^C
--- 172.24.4.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 0.403/14.731/56.028/23.845 ms
```

You can also SSH in with the key that we created during setup:

```
$ ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no \
  -i ~/id_rsa_demo cirros@172.24.4.8
```

Let's dive in and see how this gets implemented in OVN. First, the relevant parts of the NB DB for the "public" and "private" networks and the router between them:

```
$ ovn-nbctl show | abbrev
switch 2579f4 (neutron-d1ac28) (aka public)
    port provnet-d1ac28
        type: localnet
        addresses: ["unknown"]
    port ae9b52
        type: router
        router-port: lrp-ae9b52
switch 5b3d5f (neutron-c02c4d) (aka private)
    port b256dd
        type: router
        router-port: lrp-b256dd
    port f264e7
        type: router
        router-port: lrp-f264e7
    port cae25b (aka dp)
        addresses: ["fa:16:3e:c1:f5:a2 10.0.0.6 fdb0:5860:4ba8:0:f816:3eff:fec1:f5a2"]
...
router c59ad2 (neutron-9b057f) (aka router1)
    port lrp-ae9b52
        mac: "fa:16:3e:b2:d2:67"
        networks: ["172.24.4.9/24", "2001:db8::b/64"]
    port lrp-b256dd
        mac: "fa:16:3e:35:33:db"
        networks: ["fdb0:5860:4ba8::1/64"]
    port lrp-f264e7
        mac: "fa:16:3e:fc:c8:da"
        networks: ["10.0.0.1/26"]
    nat 788c6d
        external ip: "172.24.4.8"
        logical ip: "10.0.0.6"
        type: "dnat_and_snat"
    nat 80914c
        external ip: "172.24.4.9"
        logical ip: "10.0.0.0/26"
        type: "snat"
...
```

What we see is:

- VM `d` is on the "private" switch under its private IP address 10.0.0.6. The "private" switch is connected to "router1" via two router ports (one for IPv4, one for IPv6).
- The "public" switch is connected to "router1" and to the physical network via a "localnet" port.
- "router1" is in the middle between "private" and "public". In addition to the router ports that connect to these switches, it has "nat" entries that direct network address translation. The translation between floating IP address 172.24.4.8 and private address 10.0.0.6 makes perfect sense.

When the NB DB gets translated into logical flows at the southbound layer, the "nat" entries get translated into IP matches that then invoke "ct_snat" and "ct_dnat" actions. The details are intricate, but you can get some of the idea by just looking for relevant flows:

```
$ ovn-sbctl lflow-list router1 | abbrev | grep nat | grep -E '172.24.4.8'
  table=3 (lr_in_unsnat       ), priority=100  , match=(ip && ip4.dst == 172.24.4.8 &&
↪ inport == "lrp-ae9b52" && is_chassis_resident("cr-lrp-ae9b52")), action=(ct_snat;)
  table=3 (lr_in_unsnat       ), priority=50   , match=(ip && ip4.dst == 172.24.4.8),␣
↪action=(reg9[0] = 1; next;)
```

(continues on next page)

(continued from previous page)

```
  table=4 (lr_in_dnat         ), priority=100  , match=(ip && ip4.dst == 172.24.4.8 &&
↪ inport == "lrp-ae9b52" && is_chassis_resident("cr-lrp-ae9b52")), action=(ct_
↪dnat(10.0.0.6);)
  table=4 (lr_in_dnat         ), priority=50   , match=(ip && ip4.dst == 172.24.4.8),␣
↪action=(reg9[0] = 1; next;)
  table=1 (lr_out_snat        ), priority=33   , match=(ip && ip4.src == 10.0.0.6 &&␣
↪outport == "lrp-ae9b52" && is_chassis_resident("cr-lrp-ae9b52")), action=(ct_
↪snat(172.24.4.8);)
```

Let's take a look at how a packet passes through this whole gauntlet. The first two stanzas just show the packet traveling through the "public" network and being forwarded to the "router1" network:

```
$ ovn-trace public 'inport == "provnet-d1ac2896-18a7-4bca-8f46-b21e2370e5b1" && eth.
↪src == 00:01:02:03:04:05 && eth.dst == fa:16:3e:b2:d2:67 && ip4.src == 172.24.4.1 &&
↪ ip4.dst == 172.24.4.8 && ip.ttl == 64 && icmp4.type==8'
...
ingress(dp="public", inport="provnet-d1ac28")
---------------------------------------------
 0. ls_in_port_sec_l2 (northd.c:3234): inport == "provnet-d1ac28", priority 50, uuid␣
↪8d86fb06
    next;
10. ls_in_arp_rsp (northd.c:3266): inport == "provnet-d1ac28", priority 100, uuid␣
↪21313eff
    next;
13. ls_in_l2_lkup (northd.c:3571): eth.dst == fa:16:3e:b2:d2:67 && is_chassis_
↪resident("cr-lrp-ae9b52"), priority 50, uuid 7f28f51f
    outport = "ae9b52";
    output;

egress(dp="public", inport="provnet-d1ac28", outport="ae9b52")
--------------------------------------------------------------
 8. ls_out_port_sec_l2 (northd.c:3654): outport == "ae9b52", priority 50, uuid␣
↪72fea396
    output;
    /* output to "ae9b52", type "patch" */
```

In "router1", first the `ct_snat` action without an argument attempts to "un-SNAT" the packet. ovn-trace treats this as a no-op, because it doesn't have any state for tracking connections. As an alternative, it invokes `ct_dnat(10.0.0.6)` to NAT the destination IP:

```
ingress(dp="router1", inport="lrp-ae9b52")
------------------------------------------
 0. lr_in_admission (northd.c:4071): eth.dst == fa:16:3e:b2:d2:67 && inport == "lrp-
↪ae9b52" && is_chassis_resident("cr-lrp-ae9b52"), priority 50, uuid 8c6945c2
    next;
 3. lr_in_unsnat (northd.c:4591): ip && ip4.dst == 172.24.4.8 && inport == "lrp-ae9b52
↪" && is_chassis_resident("cr-lrp-ae9b52"), priority 100, uuid e922f541
    ct_snat;

ct_snat /* assuming no un-snat entry, so no change */
-----------------------------------------------------
 4. lr_in_dnat (northd.c:4649): ip && ip4.dst == 172.24.4.8 && inport == "lrp-ae9b52"␣
↪&& is_chassis_resident("cr-lrp-ae9b52"), priority 100, uuid 02f41b79
    ct_dnat(10.0.0.6);
```

Still in "router1", the routing and output steps transmit the packet to the "private" network:

```
ct_dnat(ip4.dst=10.0.0.6)
-------------------------
 5. lr_in_ip_routing (northd.c:3782): ip4.dst == 10.0.0.0/26, priority 53, uuid␣
↪86e005b0
    ip.ttl--;
    reg0 = ip4.dst;
    reg1 = 10.0.0.1;
    eth.src = fa:16:3e:fc:c8:da;
    outport = "lrp-f264e7";
    flags.loopback = 1;
    next;
 6. lr_in_arp_resolve (northd.c:5088): outport == "lrp-f264e7" && reg0 == 10.0.0.6,␣
↪priority 100, uuid 2963d67c
    eth.dst = fa:16:3e:c1:f5:a2;
    next;
 8. lr_in_arp_request (northd.c:5260): 1, priority 0, uuid eea419b7
    output;

egress(dp="router1", inport="lrp-ae9b52", outport="lrp-f264e7")
---------------------------------------------------------------
 3. lr_out_delivery (northd.c:5288): outport == "lrp-f264e7", priority 100, uuid␣
↪42dadc23
    output;
    /* output to "lrp-f264e7", type "patch" */
```

In the "private" network, the packet passes through VM d's firewall and is output to d:

```
ingress(dp="private", inport="f264e7")
--------------------------------------
 0. ls_in_port_sec_l2 (northd.c:3234): inport == "f264e7", priority 50, uuid 5b721214
    next;
 3. ls_in_pre_acl (northd.c:2624): ip && inport == "f264e7", priority 110, uuid␣
↪5bdc3209
    next;
13. ls_in_l2_lkup (northd.c:3529): eth.dst == fa:16:3e:c1:f5:a2, priority 50, uuid␣
↪7957f80f
    outport = "dp";
    output;

egress(dp="private", inport="f264e7", outport="dp")
---------------------------------------------------
 1. ls_out_pre_acl (northd.c:2648): ip, priority 100, uuid 4981c79d
    reg0[0] = 1;
    next;
 2. ls_out_pre_stateful (northd.c:2766): reg0[0] == 1, priority 100, uuid 247e02eb
    ct_next;

ct_next(ct_state=est|trk /* default (use --ct to customize) */)
---------------------------------------------------------------
 4. ls_out_acl (northd.c:2925): !ct.new && ct.est && !ct.rpl && ct_label.blocked == 0␣
↪&& (outport == "dp" && ip4 && ip4.src == 0.0.0.0/0 && icmp4), priority 2002, uuid␣
↪b860fc9f
    next;
 7. ls_out_port_sec_ip (northd.c:2364): outport == "dp" && eth.dst ==␣
↪fa:16:3e:c1:f5:a2 && ip4.dst == {255.255.255.255, 224.0.0.0/4, 10.0.0.6}, priority␣
↪90, uuid 15655a98
    next;
```

(continues on next page)

(continued from previous page)

```
 8. ls_out_port_sec_l2 (northd.c:3654): outport == "dp" && eth.dst ==
↪{fa:16:3e:c1:f5:a2}, priority 50, uuid 5916f94b
    output;
    /* output to "dp", type "" */
```

3.2.8 IPv6

OVN supports IPv6 logical routing. Let's try it out.

The first step is to add an IPv6 subnet to networks `n1` and `n2`, then attach those subnets to our router `r`. As usual, though OpenStack can assign addresses itself, we use fixed ones to make the discussion easier:

```
$ openstack subnet create --ip-version 6 --subnet-range fc11::/64 --network n1␣
↪n1subnet6
$ openstack subnet create --ip-version 6 --subnet-range fc22::/64 --network n2␣
↪n2subnet6
$ openstack router add subnet r n1subnet6
$ openstack router add subnet r n2subnet6
```

Then we add an IPv6 address to each of our VMs:

```
$ A_PORT_ID=$(openstack port list --server a -f value -c ID)
$ openstack port set --fixed-ip subnet=n1subnet6,ip-address=fc11::5 $A_PORT_ID
$ B_PORT_ID=$(openstack port list --server b -f value -c ID)
$ openstack port set --fixed-ip subnet=n1subnet6,ip-address=fc11::6 $B_PORT_ID
$ C_PORT_ID=$(openstack port list --server c -f value -c ID)
$ openstack port set --fixed-ip subnet=n2subnet6,ip-address=fc22::7 $C_PORT_ID
```

At least for me, the new IPv6 addresses didn't automatically get propagated into the VMs. To do it by hand, pull up the console for `a` and run:

```
$ sudo ip addr add fc11::5/64 dev eth0
$ sudo ip route add via fc11::1
```

Then in `b`:

```
$ sudo ip addr add fc11::6/64 dev eth0
$ sudo ip route add via fc11::1
```

Finally in `c`:

```
$ sudo ip addr add fc22::7/64 dev eth0
$ sudo ip route add via fc22::1
```

Now you should have working IPv6 routing through router `r`. The relevant parts of the NB DB look like the following. The interesting parts are the new `fc11::` and `fc22::` addresses on the ports in `n1` and `n2` and the new IPv6 router ports in `r`:

```
$ ovn-nbctl show | abbrev
...
switch f51234 (neutron-332346) (aka n2)
    port 1a8162
        type: router
        router-port: lrp-1a8162
    port 82b983
```

(continues on next page)

(continued from previous page)

```
        type: router
        router-port: lrp-82b983
    port 2e585f (aka cp)
        addresses: ["fa:16:3e:89:f2:36 10.1.2.7 fc22::7"]
switch 3eb263 (neutron-5b6baf) (aka n1)
    port ad952e
        type: router
        router-port: lrp-ad952e
    port c29d41 (aka bp)
        addresses: ["fa:16:3e:99:7a:17 10.1.1.6 fc11::6"]
    port 820c08 (aka ap)
        addresses: ["fa:16:3e:a9:4c:c7 10.1.1.5 fc11::5"]
    port 17d870
        type: router
        router-port: lrp-17d870
...
router dde06c (neutron-f88ebc) (aka r)
    port lrp-1a8162
        mac: "fa:16:3e:06:de:ad"
        networks: ["fc22::1/64"]
    port lrp-82b983
        mac: "fa:16:3e:19:9f:46"
        networks: ["10.1.2.1/24"]
    port lrp-ad952e
        mac: "fa:16:3e:ef:2f:8b"
        networks: ["fc11::1/64"]
    port lrp-17d870
        mac: "fa:16:3e:f6:e2:8f"
        networks: ["10.1.1.1/24"]
```

Try tracing a packet from `a` to `c`. The results correspond closely to those for IPv4 which we already discussed back under *Routing*:

```
$ N1SUBNET6_MAC=$(ovn-nbctl --bare --columns=mac find logical_router_port networks=\
↪"fc11::1/64\")
$ ovn-trace n1 'inport == "ap" && eth.src == '$AP_MAC' && eth.dst == '$N1SUBNET6_MAC'␣
↪&& ip6.src == fc11::5 && ip6.dst == fc22::7 && ip.ttl == 64 && icmp6.type == 8'
...
ingress(dp="n1", inport="ap")
-----------------------------
 0. ls_in_port_sec_l2 (northd.c:3234): inport == "ap" && eth.src ==
↪{fa:16:3e:a9:4c:c7}, priority 50, uuid 6dcc418a
    next;
 1. ls_in_port_sec_ip (northd.c:2390): inport == "ap" && eth.src == fa:16:3e:a9:4c:c7␣
↪&& ip6.src == {fe80::f816:3eff:fea9:4cc7, fc11::5}, priority 90, uuid 604810ea
    next;
 3. ls_in_pre_acl (northd.c:2646): ip, priority 100, uuid 46c089e6
    reg0[0] = 1;
    next;
 5. ls_in_pre_stateful (northd.c:2764): reg0[0] == 1, priority 100, uuid d1941634
    ct_next;

ct_next(ct_state=est|trk /* default (use --ct to customize) */)
---------------------------------------------------------------
 6. ls_in_acl (northd.c:2925): !ct.new && ct.est && !ct.rpl && ct_label.blocked == 0 &
↪& (inport == "ap" && ip6), priority 2002, uuid 7fdd607e
```

(continues on next page)

(continued from previous page)

```
    next;
13. ls_in_l2_lkup (northd.c:3529): eth.dst == fa:16:3e:ef:2f:8b, priority 50, uuid␣
↪e1d87fc5
    outport = "ad952e";
    output;

egress(dp="n1", inport="ap", outport="ad952e")
----------------------------------------------
 1. ls_out_pre_acl (northd.c:2626): ip && outport == "ad952e", priority 110, uuid␣
↪88f68988
    next;
 8. ls_out_port_sec_l2 (northd.c:3654): outport == "ad952e", priority 50, uuid␣
↪5935755e
    output;
    /* output to "ad952e", type "patch" */

ingress(dp="r", inport="lrp-ad952e")
------------------------------------
 0. lr_in_admission (northd.c:4071): eth.dst == fa:16:3e:ef:2f:8b && inport == "lrp-
↪ad952e", priority 50, uuid ddfeb712
    next;
 5. lr_in_ip_routing (northd.c:3782): ip6.dst == fc22::/64, priority 129, uuid␣
↪cc2130ec
    ip.ttl--;
    xxreg0 = ip6.dst;
    xxreg1 = fc22::1;
    eth.src = fa:16:3e:06:de:ad;
    outport = "lrp-1a8162";
    flags.loopback = 1;
    next;
 6. lr_in_arp_resolve (northd.c:5122): outport == "lrp-1a8162" && xxreg0 == fc22::7,␣
↪priority 100, uuid bcf75288
    eth.dst = fa:16:3e:89:f2:36;
    next;
 8. lr_in_arp_request (northd.c:5260): 1, priority 0, uuid 6dacdd82
    output;

egress(dp="r", inport="lrp-ad952e", outport="lrp-1a8162")
---------------------------------------------------------
 3. lr_out_delivery (northd.c:5288): outport == "lrp-1a8162", priority 100, uuid␣
↪5260dfc5
    output;
    /* output to "lrp-1a8162", type "patch" */

ingress(dp="n2", inport="1a8162")
---------------------------------
 0. ls_in_port_sec_l2 (northd.c:3234): inport == "1a8162", priority 50, uuid 10957d1b
    next;
 3. ls_in_pre_acl (northd.c:2624): ip && inport == "1a8162", priority 110, uuid␣
↪a27ebd00
    next;
13. ls_in_l2_lkup (northd.c:3529): eth.dst == fa:16:3e:89:f2:36, priority 50, uuid␣
↪dcafb3e9
    outport = "cp";
    output;

egress(dp="n2", inport="1a8162", outport="cp")
```

(continues on next page)

(continued from previous page)

```
------------------------------------------
 1. ls_out_pre_acl (northd.c:2648): ip, priority 100, uuid cd9cfa74
    reg0[0] = 1;
    next;
 2. ls_out_pre_stateful (northd.c:2766): reg0[0] == 1, priority 100, uuid 9e8e22c5
    ct_next;

ct_next(ct_state=est|trk /* default (use --ct to customize) */)
---------------------------------------------------------------
 4. ls_out_acl (northd.c:2925): !ct.new && ct.est && !ct.rpl && ct_label.blocked == 0␣
↪&& (outport == "cp" && ip6 && ip6.src == $as_ip6_0fc1b6cf_f925_49e6_8f00_
↪6dd13beca9dc), priority 2002, uuid 12fc96f9
    next;
 7. ls_out_port_sec_ip (northd.c:2390): outport == "cp" && eth.dst ==␣
↪fa:16:3e:89:f2:36 && ip6.dst == {fe80::f816:3eff:fe89:f236, ff00::/8, fc22::7},␣
↪priority 90, uuid c622596a
    next;
 8. ls_out_port_sec_l2 (northd.c:3654): outport == "cp" && eth.dst ==
↪{fa:16:3e:89:f2:36}, priority 50, uuid 0242cdc3
    output;
    /* output to "cp", type "" */
```

3.2.9 ACLs

Let's explore how ACLs work in OpenStack and OVN. In OpenStack, ACL rules are part of "security groups", which are "default deny", that is, packets are not allowed by default and the rules added to security groups serve to allow different classes of packets. The default group (named "default") that is assigned to each of our VMs so far allows all traffic from our other VMs, which isn't very interesting for testing. So, let's create a new security group, which we'll name "custom", add rules to it that allow incoming SSH and ICMP traffic, and apply this security group to VM `c`:

```
$ openstack security group create custom
$ openstack security group rule create --dst-port 22 custom
$ openstack security group rule create --protocol icmp custom
$ openstack server remove security group c default
$ openstack server add security group c custom
```

Now we can do some experiments to test security groups. From the console on `a` or `b`, it should now be possible to "ping" `c` or to SSH to it, but attempts to initiate connections on other ports should be blocked. (You can try to connect on another port with `ssh -p PORT IP` or `nc IP PORT -vv`.) Connection attempts should time out rather than receive the "connection refused" or "connection reset" error that you would see between `a` and `b`.

It's also possible to test ACLs via ovn-trace, with one new wrinkle. ovn-trace can't simulate connection tracking state in the network, so by default it assumes that every packet represents an established connection. That's good enough for what we've been doing so far, but for checking properties of security groups we want to look at more detail.

If you look back at the VM-to-VM traces we've done until now, you can see that they execute two `ct_next` actions:

- The first of these is for the packet passing outward through the source VM's firewall. We can tell ovn-trace to treat the packet as starting a new connection or adding to an established connection by adding a `--ct` option: `--ct new` or `--ct est`, respectively. The latter is the default and therefore what we've been using so far. We can also use `--ct est,rpl`, which in addition to `--ct est` means that the connection was initiated by the destination VM rather than by the VM sending this packet.
- The second is for the packet passing inward through the destination VM's firewall. For this one, it makes sense to tell ovn-trace that the packet is starting a new connection, with `--ct new`, or that it is a packet sent in reply to a connection established by the destination VM, with `--ct est,rpl`.

ovn-trace uses the `--ct` options in order, so if we want to override the second `ct_next` behavior we have to specify two options.

Another useful ovn-trace option for this testing is `--minimal`, which reduces the amount of output. In this case we're really just interested in finding out whether the packet reaches the destination VM, that is, whether there's an eventual `output` action to `c`, so `--minimal` works fine and the output is easier to read.

Try a few traces. For example:

First, obtain the mac address of logical router's IPv4 interface on `n1`:

```
$ N1SUBNET4_MAC=$(ovn-nbctl --bare --columns=mac \
  find logical_router_port networks=\"10.1.1.1/24\")
```

- VM `a` initiates a new SSH connection to `c`:

```
$ ovn-trace --ct new --ct new --minimal n1 'inport == "ap" &&
  eth.src == '$AP_MAC' && eth.dst == '$N1SUBNET4_MAC' &&
  ip4.src == 10.1.1.5 && ip4.dst == 10.1.2.7 && ip.ttl == 64 &&
  tcp.dst == 22'
...
ct_next(ct_state=new|trk) {
    ip.ttl--;
    eth.src = fa:16:3e:19:9f:46;
    eth.dst = fa:16:3e:89:f2:36;
    ct_next(ct_state=new|trk) {
        output("cp");
    };
};
```

  This succeeds, as you can see since there is an `output` action.

- VM `a` initiates a new Telnet connection to `c`:

```
$ ovn-trace --ct new --ct new --minimal n1 'inport == "ap" &&
  eth.src == '$AP_MAC' && eth.dst == '$N1SUBNET4_MAC' &&
  ip4.src == 10.1.1.5 && ip4.dst == 10.1.2.7 && ip.ttl == 64 &&
  tcp.dst == 23'
ct_next(ct_state=new|trk) {
    ip.ttl--;
    eth.src = fa:16:3e:19:9f:46;
    eth.dst = fa:16:3e:89:f2:36;
    ct_next(ct_state=new|trk);
};
```

  This fails, as you can see from the lack of an `output` action.

- VM `a` replies to a packet that is part of a Telnet connection originally initiated by `c`:

```
$ ovn-trace --ct est,rpl --ct est,rpl --minimal n1 'inport == "ap" &&
  eth.src == '$AP_MAC' && eth.dst == '$N1SUBNET4_MAC' &&
  ip4.src == 10.1.1.5 && ip4.dst == 10.1.2.7 && ip.ttl == 64 &&
  tcp.dst == 23'
...
ct_next(ct_state=est|rpl|trk) {
    ip.ttl--;
    eth.src = fa:16:3e:19:9f:46;
    eth.dst = fa:16:3e:89:f2:36;
    ct_next(ct_state=est|rpl|trk) {
```

(continues on next page)

(continued from previous page)

```
        output("cp");
    };
};
```

This succeeds, as you can see from the `output` action, since traffic received in reply to an outgoing connection is always allowed.

3.2.10 DHCP

As a final demonstration of the OVN architecture, let's examine the DHCP implementation. Like switching, routing, and NAT, the OVN implementation of DHCP involves configuration in the NB DB and logical flows in the SB DB.

Let's look at the DHCP support for `a`'s port `ap`. The port's Logical_Switch_Port record shows that `ap` has DHCPv4 options:

```
$ ovn-nbctl list logical_switch_port ap | abbrev
_uuid               : ef17e5
addresses           : ["fa:16:3e:a9:4c:c7 10.1.1.5 fc11::5"]
dhcpv4_options      : 165974
dhcpv6_options      : 26f7cd
dynamic_addresses   : []
enabled             : true
external_ids        : {"neutron:port_name"=ap}
name                : "820c08"
options             : {}
parent_name         : []
port_security       : ["fa:16:3e:a9:4c:c7 10.1.1.5 fc11::5"]
tag                 : []
tag_request         : []
type                : ""
up                  : true
```

We can then list them either by UUID or, more easily, by port name:

```
$ ovn-nbctl list dhcp_options ap | abbrev
_uuid               : 165974
cidr                : "10.1.1.0/24"
external_ids        : {subnet_id="5e67e7"}
options             : {lease_time="43200", mtu="1442", router="10.1.1.1", server_id=
↪"10.1.1.1", server_mac="fa:16:3e:bb:94:72"}
```

These options show the basic DHCP configuration for the subnet. They do not include the IP address itself, which comes from the Logical_Switch_Port record. This allows a whole Neutron subnet to share a single DHCP_Options record. You can see this sharing in action, if you like, by listing the record for port `bp`, which is on the same subnet as `ap`, and see that it is the same record as before:

```
$ ovn-nbctl list dhcp_options bp | abbrev
_uuid               : 165974
cidr                : "10.1.1.0/24"
external_ids        : {subnet_id="5e67e7"}
options             : {lease_time="43200", mtu="1442", router="10.1.1.1", server_id=
↪"10.1.1.1", server_mac="fa:16:3e:bb:94:72"}
```

You can take another look at the southbound flow table if you like, but the best demonstration is to trace a DHCP packet. The following is a trace of a DHCP request inbound from `ap`. The first part is just the usual travel through the firewall:

```
$ ovn-trace n1 'inport == "ap" && eth.src == '$AP_MAC' && eth.dst ==␣
↪ff:ff:ff:ff:ff:ff && ip4.dst == 255.255.255.255 && udp.src == 68 && udp.dst == 67 &&
↪ ip.ttl == 1'
...
ingress(dp="n1", inport="ap")
-----------------------------
 0. ls_in_port_sec_l2 (northd.c:3234): inport == "ap" && eth.src ==
↪{fa:16:3e:a9:4c:c7}, priority 50, uuid 6dcc418a
    next;
 1. ls_in_port_sec_ip (northd.c:2325): inport == "ap" && eth.src == fa:16:3e:a9:4c:c7␣
↪&& ip4.src == 0.0.0.0 && ip4.dst == 255.255.255.255 && udp.src == 68 && udp.dst ==␣
↪67, priority 90, uuid e46bed6f
    next;
 3. ls_in_pre_acl (northd.c:2646): ip, priority 100, uuid 46c089e6
    reg0[0] = 1;
    next;
 5. ls_in_pre_stateful (northd.c:2764): reg0[0] == 1, priority 100, uuid d1941634
    ct_next;
```

The next part is the new part. First, an ACL in table 6 allows a DHCP request to pass through. In table 11, the special `put_dhcp_opts` action replaces a DHCPDISCOVER or DHCPREQUEST packet by a reply. Table 12 flips the packet's source and destination and sends it back the way it came in:

```
 6. ls_in_acl (northd.c:2925): !ct.new && ct.est && !ct.rpl && ct_label.blocked == 0 &
↪& (inport == "ap" && ip4 && ip4.dst == {255.255.255.255, 10.1.1.0/24} && udp && udp.
↪src == 68 && udp.dst == 67), priority 2002, uuid 9c90245d
    next;
11. ls_in_dhcp_options (northd.c:3409): inport == "ap" && eth.src ==␣
↪fa:16:3e:a9:4c:c7 && ip4.src == 0.0.0.0 && ip4.dst == 255.255.255.255 && udp.src ==␣
↪68 && udp.dst == 67, priority 100, uuid 8d63f29c
    reg0[3] = put_dhcp_opts(offerip = 10.1.1.5, lease_time = 43200, mtu = 1442,␣
↪netmask = 255.255.255.0, router = 10.1.1.1, server_id = 10.1.1.1);
    /* We assume that this packet is DHCPDISCOVER or DHCPREQUEST. */
    next;
12. ls_in_dhcp_response (northd.c:3438): inport == "ap" && eth.src ==␣
↪fa:16:3e:a9:4c:c7 && ip4 && udp.src == 68 && udp.dst == 67 && reg0[3], priority 100,
↪ uuid 995eeaa9
    eth.dst = eth.src;
    eth.src = fa:16:3e:bb:94:72;
    ip4.dst = 10.1.1.5;
    ip4.src = 10.1.1.1;
    udp.src = 67;
    udp.dst = 68;
    outport = inport;
    flags.loopback = 1;
    output;
```

Then the last part is just traveling back through the firewall to VM `a`:

```
egress(dp="n1", inport="ap", outport="ap")
------------------------------------------
 1. ls_out_pre_acl (northd.c:2648): ip, priority 100, uuid 3752b746
    reg0[0] = 1;
    next;
 2. ls_out_pre_stateful (northd.c:2766): reg0[0] == 1, priority 100, uuid 0c066ea1
    ct_next;
```

(continues on next page)

(continued from previous page)

```
ct_next(ct_state=est|trk /* default (use --ct to customize) */)
---------------------------------------------------------------
 4. ls_out_acl (northd.c:3008): outport == "ap" && eth.src == fa:16:3e:bb:94:72 &&␣
↪ip4.src == 10.1.1.1 && udp && udp.src == 67 && udp.dst == 68, priority 34000, uuid␣
↪0b383e77
    ct_commit;
    next;
 7. ls_out_port_sec_ip (northd.c:2364): outport == "ap" && eth.dst ==␣
↪fa:16:3e:a9:4c:c7 && ip4.dst == {255.255.255.255, 224.0.0.0/4, 10.1.1.5}, priority␣
↪90, uuid 7b8cbcd5
    next;
 8. ls_out_port_sec_l2 (northd.c:3654): outport == "ap" && eth.dst ==
↪{fa:16:3e:a9:4c:c7}, priority 50, uuid b874ece8
    output;
    /* output to "ap", type "" */
```

3.2.11 Further Directions

We've looked at a fair bit of how OVN works and how it interacts with OpenStack. If you still have some interest, then you might want to explore some of these directions:

- Adding more than one hypervisor ("compute node", in OpenStack parlance). OVN connects compute nodes by tunneling packets with the STT or Geneve protocols. OVN scales to 1000 compute nodes or more, but two compute nodes demonstrate the principle. All of the tools and techniques we demonstrated also work with multiple compute nodes.
- Container support. OVN supports seamlessly connecting VMs to containers, whether the containers are hosted on "bare metal" or nested inside VMs. OpenStack support for containers, however, is still evolving, and too difficult to incorporate into the tutorial at this point.
- Other kinds of gateways. In addition to floating IPs with NAT, OVN supports directly attaching VMs to a physical network and connecting logical switches to VTEP hardware.

3.3 OVN Role-Based Access Control (RBAC) Tutorial

This document provides a step-by-step guide for setting up role-based access control (RBAC) in OVN. In OVN, hypervisors (chassis) read and write the southbound database to do configuration. Without restricting hypervisor's access to the southbound database, a compromised hypervisor might disrupt the entire OVN deployment by corrupting the database. RBAC ensures that each hypervisor can only modify its own data and thus improves the security of OVN. More details about the RBAC design can be found in `ovn-architecture`(7) manpage.

This document assumes OVN is installed in your system and runs normally.

3.3.1 Generating Certificates and Keys

In the OVN RBAC deployment, ovn-controller connects to the southbound database via SSL connection. The southbound database uses CA-signed certificate to authenticate ovn-controller.

Suppose there are three machines in your deployment. *machine_1* runs *chassis_1* and has IP address *machine_1-ip*. *machine_2* runs *chassis_2* and has IP address *machine_2-ip*. *machine_3* hosts southbound database and has IP address *machine_3-ip*. *machine_3* also hosts public key infrastructure (PKI).

1. Initiate PKI.

   In *machine_3*:

```
$ ovs-pki init
```

2. Generate southbound database's certificate request. Sign the certificate request with the CA key.

   In *machine_3*:

```
$ ovs-pki req -u sbdb
$ ovs-pki sign sbdb switch
```

3. Generate chassis certificate requests. Copy the certificate requests to *machine_3*.

   In *machine_1*:

```
$ ovs-pki req -u chassis_1
$ scp chassis_1-req.pem \
              machine_3@machine_3-ip:/path/to/chassis_1-req.pem
```

   In *machine_2*:

```
$ ovs-pki req -u chassis_2
$ scp chassis_2-req.pem \
              machine_3@machine_3-ip:/path/to/chassis_2-req.pem
```

Note: chassis_1 must be the same string as `external_ids:system-id` in the Open_vSwitch table (the chassis name) of machine_1. Same applies for chassis_2.

---

4. Sign the chassis certificate requests with the CA key. Copy *chassis_1*'s signed certificate and the CA certificate to *machine_1*. Copy *chassis_2*'s signed certificate and the CA certificate to *machine_2*.

   In *machine_3*:

```
$ ovs-pki sign chassis_1 switch
$ ovs-pki sign chassis_2 switch
$ scp chassis_1-cert.pem \
              machine_1@machine_1-ip:/path/to/chassis_1-cert.pem
$ scp /var/lib/openvswitch/pki/switchca/cacert.pem \
              machine_1@machine_1-ip:/path/to/cacert.pem
$ scp chassis_2-cert.pem \
              machine_2@machine_2-ip:/path/to/chassis_2-cert.pem
$ scp /var/lib/openvswitch/pki/switchca/cacert.pem \
              machine_2@machine_2-ip:/path/to/cacert.pem
```

3.3.2 Configuring RBAC

1. Set certificate, private key, and CA certificate for the southbound database. Configure the southbound database to listen on SSL connection and enforce role-based access control.

   In *machine_3*:

```
$ ovn-sbctl set-ssl /path/to/sbdb-privkey.pem \
                    /path/to/sbdb-cert.pem /path/to/cacert.pem
$ ovn-sbctl set-connection role=ovn-controller pssl:6642
```

2. Set certificate, private key, and CA certificate for *chassis_1* and *chassis_2*. Configure *chassis_1* and *chassis_2* to connect southbound database via SSL.

   In *machine_1*:

```
$ ovs-vsctl set-ssl /path/to/chassis_1-privkey.pem \
              /path/to/chassis_1-cert.pem /path/to/cacert.pem
$ ovs-vsctl set open_vswitch . \
              external_ids:ovn-remote=ssl:machine_3-ip:6642
```

   In *machine_2*:

```
$ ovs-vsctl set-ssl /path/to/chassis_2-privkey.pem \
              /path/to/chassis_2-cert.pem /path/to/cacert.pem
$ ovs-vsctl set open_vswitch . \
              external_ids:ovn-remote=ssl:machine_3-ip:6642
```

The OVN central control daemon and RBAC

The OVN central control daemon (*ovn-northd*) needs full write access to the southbound database. When you have one machine hosting the central components, *ovn-northd* can talk to the databases through a local unix socket, bypassing the *ovn-controller* RBAC configured for the listener at port '6642'. However, if you want to deploy multiple machines for hosting the central components, *ovn-northd* will require a remote connection to all of them.

1. Configure the southbound database with a second SSL listener on a separate port without RBAC enabled for use by *ovn-northd*.

   In *machine_3*:

```
$ ovn-sbctl -- --id=@conn_uuid create Connection \
    target="pssl\:16642" \
    -- add  SB_Global . connections=@conn_uuid
```

Note: Care should be taken to restrict access to the above mentioned port so that only trusted machines can connect to it.

3.4 OVN IPsec Tutorial

This document provides a step-by-step guide for encrypting tunnel traffic with IPsec in Open Virtual Network (OVN). OVN tunnel traffic is transported by physical routers and switches. These physical devices could be untrusted (devices in public network) or might be compromised. Enabling IPsec encryption for the tunnel traffic can prevent the traffic data from being monitored and manipulated. More details about the OVN IPsec design can be found in `ovn-architecture`(7) manpage.

This document assumes OVN is installed in your system and runs normally. Also, you need to install OVS IPsec packages in each chassis (refer to Open vSwitch documentation on ipsec).

3.4.1 Generating Certificates and Keys

OVN chassis uses CA-signed certificate to authenticate peer chassis for building IPsec tunnel. If you have enabled Role-Based Access Control (RBAC) in OVN, you can use the RBAC SSL certificates and keys to set up OVN IPsec. Or you can generate separate certificates and keys with `ovs-pki` (refer to *Generating Certificates and Keys*).

Note: OVN IPsec requires x.509 version 3 certificate with the subjectAltName DNS field setting the same string as the common name (CN) field. CN should be set as the chassis name. `ovs-pki` in Open vSwitch 2.10.90 and later generates such certificates. Please generate compatible certificates if you use another PKI tool, or an older version of `ovs-pki`, to manage certificates.

3.4.2 Configuring OVN IPsec

You need to install the CA certificate, chassis certificate and private key in each chassis. Use the following command:

```
$ ovs-vsctl set Open_vSwitch . \
        other_config:certificate=/path/to/chassis-cert.pem \
        other_config:private_key=/path/to/chassis-privkey.pem \
        other_config:ca_cert=/path/to/cacert.pem
```

3.4.3 Enabling OVN IPsec

To enable OVN IPsec, set `ipsec` column in `NB_Global` table of the northbound database to true:

```
$ ovn-nbctl set nb_global . ipsec=true
```

With OVN IPsec enabled, all tunnel traffic in OVN will be encrypted with IPsec. To disable it, set `ipsec` column in `NB_Global` table of the northbound database to false:

```
$ ovn-nbctl set nb_global . ipsec=false
```

Note: On Fedora, RHEL and CentOS, you may need to install firewall rules to allow ESP and IKE traffic:

```
# systemctl start firewalld
# firewall-cmd --add-service ipsec
```

Or to make permanent:

```
# systemctl enable firewalld
# firewall-cmd --permanent --add-service ipsec
```

3.4.4 Enforcing IPsec NAT-T UDP encapsulation

In specific situations, it may be required to enforce NAT-T (RFC3948) UDP encapsulation unconditionally and to bypass the normal NAT detection mechanism. For example, this may be required in environments where firewalls drop ESP traffic, but where NAT-T detection (RFC3947) fails because packets otherwise are not subject to NAT. In such scenarios, UDP encapsulation can be enforced with the following.

For libreswan backends:

```
$ ovn-nbctl set nb_global . options:ipsec_encapsulation=true
```

For strongswan backends:

```
$ ovn-nbctl set nb_global . options:ipsec_forceencaps=true
```

Note: Support for this feature is only availably when OVN is used together with OVS releases that accept IPsec custom tunnel options.

3.4.5 Troubleshooting

The `ovs-monitor-ipsec` daemon in each chassis manages and monitors the IPsec tunnel state. Use the following `ovs-appctl` command to view `ovs-monitor-ipsec` internal representation of tunnel configuration:

```
$ ovs-appctl -t ovs-monitor-ipsec tunnels/show
```

If there is a misconfiguration, then `ovs-appctl` should indicate why. For example:

```
Interface name: ovn-host_2-0 v1 (CONFIGURED) <--- Should be set
                                          to CONFIGURED. Otherwise,
                                          error message will be
                                          provided
Tunnel Type:    geneve
Remote IP:      2.2.2.2
SKB mark:       None
Local cert:     /path/to/chassis-cert.pem
Local name:     host_1
Local key:      /path/to/chassis-privkey.pem
Remote cert:    None
Remote name:    host_2
CA cert:        /path/to/cacert.pem
PSK:            None
Custom Options: {'encapsulation': 'yes'} <---- Whether NAT-T is enforced
Ofport:         2          <--- Whether ovs-vswitchd has assigned Ofport
                              number to this Tunnel Port
CFM state:      Disabled     <--- Whether CFM declared this tunnel healthy
Kernel policies installed:
...                          <--- IPsec policies for this OVS tunnel in
                                  Linux Kernel installed by strongSwan
Kernel security associations installed:
...                          <--- IPsec security associations for this OVS
                                  tunnel in Linux Kernel installed by
                                  strongswan
IPsec connections that are active:
...                          <--- IPsec "connections" for this OVS
                                  tunnel
```

If you don't see any active connections, try to run the following command to refresh the `ovs-monitor-ipsec` daemon:

```
$ ovs-appctl -t ovs-monitor-ipsec refresh
```

You can also check the logs of the `ovs-monitor-ipsec` daemon and the IKE daemon to locate issues. `ovs-monitor-ipsec` outputs log messages to `/var/log/openvswitch/ovs-monitor-ipsec.log`.

3.4.6 Bug Reporting

If you think you may have found a bug with security implications, like

1. IPsec protected tunnel accepted packets that came unencrypted; OR
2. IPsec protected tunnel allowed packets to leave unencrypted;

Then report such bugs according to *OVN's Security Process*.

If bug does not have security implications, then report it according to instructions in *Reporting Bugs in OVN*.

If you have suggestions to improve this tutorial, please send a email to ovs-discuss@openvswitch.org.

3.5 OVN Interconnection

This document provides a guide for interconnecting multiple OVN deployements with OVN managed tunneling. More details about the OVN Interconnectiong design can be found in `ovn-architecture`(7) manpage.

This document assumes two or more OVN deployments are setup and runs normally, possibly at different data-centers, and the gateway chassises of each OVN are with IP addresses that are reachable between each other.

3.5.1 Setup Interconnection Databases

To interconnect different OVNs, you need to create global OVSDB databases that store interconnection data. The databases can be setup on any nodes that are accessible from all the central nodes of each OVN deployment. It is recommended that the global databases are setup with HA, with nodes in different avaialbility zones, to avoid single point of failure.

1. Install OVN packages on each global database node.
2. Start OVN IC-NB and IC-SB databases.

   On each global database node

   ```
   $ ovn-ctl [options] start_ic_ovsdb
   ```

   Options depends on the HA mode you use. To start standalone mode with TCP connections, use

   ```
   $ ovn-ctl --db-ic-nb-create-insecure-remote=yes \
             --db-ic-sb-create-insecure-remote=yes start_ic_ovsdb
   ```

   This command starts IC database servers that accept both unix socket and TCP connections. For other modes, see more details with

   ```
   $ ovn-ctl --help
   ```

3.5.2 Register OVN to Interconnection Databases

For each OVN deployment, set an availability zone name

```
$ ovn-nbctl set NB_Global . name=<availability zone name>
```

The name should be unique across all OVN deployments, e.g. ovn-east, ovn-west, etc.

For each OVN deployment, start the `ovn-ic` daemon on central nodes

```
$ ovn-ctl --ovn-ic-nb-db=<IC-NB> --ovn-ic-sb-db=<IC-SB> \
          --ovn-northd-nb-db=<NB> --ovn-northd-sb-db=<SB> [more options] start_ic
```

An example of <IC-NB> is tcp:<global db hostname>:6645, or for clustered DB: tcp:<node1>:6645,tcp:<node2>:6645,tcp:<node3>:6645. <IC-SB> is similar, but usually with a different port number, typically, 6646.

For <NB> and <SB>, use same connection methods as for starting northd.

Verify each OVN registration from global IC-SB database, using ovn-ic-sbctl, either on a global DB node or other nodes but with property DB connection method specified in options

```
$ ovn-ic-sbctl show
```

3.5.3 Configure Gateways

For each OVN deployment, specify some chassises as interconnection gateways. The number of gateways you need depends on the scale and bandwidth you need for the traffic between the OVN deployments.

For a node to work as an interconnection gateway, it must firstly be installed and configured as a regular OVN chassis, with OVS and ovn-controller running. To make a chassis as an interconnection gateway, simply run the command on the chassis

```
$ ovs-vsctl set open_vswitch . external_ids:ovn-is-interconn=true
```

After configuring gateways, verify from the global IC-SB database

```
$ ovn-ic-sbctl show
```

3.5.4 Create Transit Logical Switches

Transit Logical Switches, or Transit Switches, are virtual switches for connecting logical routers in different OVN setups.

```
$ ovn-ic-nbctl ts-add <name>
```

After creating a transit switch, it can be seen from each OVN deployment's Northbound database, which can be seen using

```
$ ovn-nbctl find logical_switch other_config:interconn-ts=<name>
```

You will also see it with simply ovn-nbctl ls-list.

If there are multiple tenants that require traffic being isolated from each other, then multiple transit switches can be created accordingly.

3.5.5 Connect Logical Routers to Transit Switches

Connect logical routers from each OVN deployment to the desired transit switches just as if they are regular logical switches, which includes below steps (from each OVN, for each logical router you want to connect).

Assume a transit switch named ts1 is already created in IC-NB and a logical router lr1 created in current OVN deployment.

1. Create a logical router port.

```
$ ovn-nbctl lrp-add lr1 lrp-lr1-ts1 aa:aa:aa:aa:aa:01 169.254.100.1/24
```

   (The mac and IP are examples.)

2. Create a logical switch port on the transit switch and peer with the logical router port.

```
$ ovn-nbctl lsp-add ts1 lsp-ts1-lr1 -- \
        lsp-set-addresses lsp-ts1-lr1 router -- \
        lsp-set-type lsp-ts1-lr1 router -- \
        lsp-set-options lsp-ts1-lr1 router-port=lrp-lr1-ts1
```

3. Assign gateway(s) for the logical router port.

```
$ ovn-nbctl lrp-set-gateway-chassis lrp-lr1-ts1 <gateway name> [priority]
```

   Optionally, you can assign more gateways and specify priorities, to achieve HA, just as usual for a distributed gateway port.

Similarly in another OVN deployment, you can connect a logical router (e.g. lr2) to the same transit switch the same way, with a different IP (e.g. 169.254.100.2) on the same subnet.

The ports connected to transit switches will be automatically populated to `IC-SB` database, which can be verified by

```
$ ovn-ic-sbctl show
```

3.5.6 Create Static Routes

Now that you have all the physical and logical topologies ready, simply create static routes between the OVN deployments so that packets can be forwarded by the logical routers through transit switches to the remote OVN.

For example, in ovn-east, there are workloads using 10.0.1.0/24 under lr1, and in ovn-west, there are workloads using 10.0.2.0/24 under lr2.

In ovn-east, add below route

```
$ ovn-nbctl lr-route-add lr1 10.0.2.0/24 169.254.100.2
```

In ovn-west, add below route

```
$ ovn-nbctl lr-route-add lr2 10.0.1.0/24 169.254.100.1
```

Now the traffic should be able to go through between the workloads through tunnels crossing gateway nodes of ovn-east and ovn-west.

3.5.7 Route Advertisement

Alternatively, you can avoid the above manual static route configuration by enabling route advertisement and learning on each OVN deployment

```
$ ovn-nbctl set NB_Global . options:ic-route-adv=true \
                            options:ic-route-learn=true
```

With this setting, the above routes will be automatically learned and configured in Northbound DB in each deployment. For example, in ovn-east, you will see the route

```
$ ovn-nbctl lr-route-list lr1
IPv4 Routes
             10.0.2.0/24             169.254.100.2 dst-ip (learned)
```

In ovn-west you will see

```
$ ovn-nbctl lr-route-list lr2
IPv4 Routes
             10.0.1.0/24             169.254.100.1 dst-ip (learned)
```

Static routes configured in the routers can be advertised and learned as well. For more details of router advertisement and its configure options, please see <code>ovn-nb</code>(5).

3.6 Adding a new OVN feature to the DDlog version of ovn-northd

This document describes the usual steps an OVN developer should go through when adding a new feature to `ovn-northd-ddlog`. In order to make things less abstract we will use the IP Multicast `ovn-northd-ddlog` implementation as an example. Even though the document is structured as a tutorial there might still exist feature-specific aspects that are not covered here.

3.6.1 Overview

DDlog is a dataflow system: it receives data from a data source (a set of "input relations"), processes it through "intermediate relations" according to the rules specified in the DDlog program, and sends the processed "output relations" to a data sink. In OVN, the input relations primarily come from the OVN Northbound database and the output relations primarily go to the OVN Southbound database. The process looks like this:

```
from NBDB  +-----------+   +------------------+   +------------+  to SBDB
---------->|Input rels|-->|Intermediate rels|-->|Output rels|---------->
           +-----------+   +------------------+   +------------+
```

Adding a new feature to `ovn-northd-ddlog` usually involves the following steps:

1. Update northbound and/or southbound OVSDB schemas.
2. Configure DDlog/OVSDB bindings.
3. Define intermediate DDlog relations and rules to compute them.
4. Write rules to update output relations.
5. Generate `Logical_Flow``s and/or other forwarding records (e.g., ``Multicast_Group`) that will control the dataplane operations.

3.6.2 Update NB and/or SB OVSDB schemas

This step is no different from the normal development flow in C.

Most of the times a developer chooses between two ways of configuring a new feature:

1. Adding a set of columns to tables in the NB and/or SB database (or adding key-value pairs to existing columns).
2. Adding new tables to the NB and/or SB database.

Looking at IP Multicast, there are two `OVN Northbound` tables where configuration information is stored:

- `Logical_Switch`, column `other_config`, keys `mcast_*`.
- `Logical_Router`, column `options`, keys `mcast_*`.

These tables become inputs to the DDlog pipeline.

In addition we add a new table `IP_Multicast` to the SB database. DDlog will update this table, that is, `IP_Multicast` receives output from the above pipeline.

3.6.3 Configuring DDlog/OVSDB bindings

Configuring `northd/automake.mk`

The OVN build process uses DDlog's `ovsdb2ddlog` utility to parse `ovn-nb.ovsschema` and `ovn-sb.ovsschema` and then automatically populate `OVN_Northbound.dl` and `OVN_Southbound.dl`. For each OVN Northbound and Southbound table, it generates one or more corresponding DDlog relations.

We need to supply `ovsdb2ddlog` with some information that it can't infer from the OVSDB schemas. This information must be specified as `ovsdb2ddlog` arguments, which are read from `northd/ovn-nb.dlopts` and `northd/ovn-sb.dlopts`.

The main choice for each new table is whether it is used for output. Output tables can also be used for input, but the converse is not true. If the table is used for output at all, we add `-o <table>` to the option file. Our new table `IP_Multicast` is an output table, so we add `-o IP_Multicast` to `ovn-sb.dlopts`.

For input-only tables, `ovsdb2ddlog` generates a DDlog input relation with the same name. For output tables, it generates this table plus an output relation named `Out_<table>`. Thus, `OVN_Southbound.dl` has two relations for `IP_Multicast`:

```
input relation IP_Multicast (
    _uuid: uuid,
    datapath: string,
    enabled: Set<bool>,
    querier: Set<bool>
)
output relation Out_IP_Multicast (
    _uuid: uuid,
    datapath: string,
    enabled: Set<bool>,
    querier: Set<bool>
)
```

For an output table, consider whether only some of the columns are used for output, that is, some of the columns are effectively input-only. This is common in OVN for OVSDB columns that are managed externally (e.g. by a CMS). For each input-only column, we add `--ro <table>.<column>`. Alternatively, if most of the columns are input-only but a few are output columns, add `--rw <table>.<column>` for each of the output columns. In our case, all of the columns are used for output, so we do not need to add anything.

Finally, in some cases `ovn-northd-ddlog` shouldn't change values in . One such case is the `seq_no` column in the `IP_Multicast` table. To do that we need to instruct `ovsdb2ddlog` to treat the column as read-only by using the `--ro` switch.

`ovsdb2ddlog` generates a number of additional DDlog relations, for use by auto-generated OVSDB adapter logic. These are irrelevant to most DDLog developers, although sometimes they can be handy for debugging. See the *appendix* for details.

3.6.4 Define intermediate DDlog relations and rules to compute them.

Obviously there will be a one-to-one relationship between logical switches/routers and IP multicast configuration. One way to represent this relationship is to create multicast configuration DDlog relations to be referenced by `&Switch` and `&Router` DDlog records:

```
/* IP Multicast per switch configuration. */
relation &McastSwitchCfg(
    datapath      : uuid,
    enabled       : bool,
    querier       : bool
}

&McastSwitchCfg(
        .datapath = ls_uuid,
        .enabled  = map_get_bool_def(other_config, "mcast_snoop", false),
        .querier  = map_get_bool_def(other_config, "mcast_querier", true)) :-
    nb.Logical_Switch(._uuid        = ls_uuid,
                      .other_config = other_config).
```

Then reference these relations in `&Switch` and `&Router`. For example, in `lswitch.dl`, the `&Switch` relation definition now contains:

```
relation &Switch(
    ls:                 nb.Logical_Switch,
    [...]
    mcast_cfg:          Ref<McastSwitchCfg>
)
```

And is populated by the following rule which references the correct `McastSwitchCfg` based on the logical switch uuid:

```
&Switch(.ls         = ls,
        [...]
        .mcast_cfg = mcast_cfg) :-
    nb.Logical_Switch[ls],
    [...]
    mcast_cfg in &McastSwitchCfg(.datapath = ls._uuid).
```

Build state based on information dynamically updated by `ovn-controller`

Some OVN features rely on information learned by `ovn-controller` to generate `Logical_Flow` or other records that control the dataplane. In case of IP Multicast, `ovn-controller` uses IGMP to learn multicast groups that are joined by hosts.

Each `ovn-controller` maintains its own set of records to avoid ownership and concurrency with other controllers. If two hosts that are connected to the same logical switch but reside on different hypervisors (different `ovn-controller` processes) join the same multicast group G, each of the controllers will create an `IGMP_Group` record in the `OVN Southbound` database which will contain a set of ports to which the interested hosts are connected.

At this point `ovn-northd-ddlog` needs to aggregate the per-chassis IGMP records to generate a single `Logical_Flow` for group G. Moreover, the ports on which the hosts are connected are represented as references to `Port_Binding` records in the database. These also need to be translated to `&SwitchPort` DDlog relations. The corresponding DDlog operations that need to be performed are:

- Flatten the <IGMP group, ports> mapping in order to be able to do the translation from Port_Binding to &SwitchPort. For each IGMP_Group record in the OVN Southbound database generate an individual record of type IgmpSwitchGroupPort for each Port_Binding in the set of ports that joined the group. Also, translate the Port_Binding uuid to the corresponding Logical_Switch_Port uuid:

```
relation IgmpSwitchGroupPort(
    address: string,
    switch : Ref<Switch>,
    port   : uuid
)

IgmpSwitchGroupPort(address, switch, lsp_uuid) :-
    sb::IGMP_Group(.address = address, .datapath = igmp_dp_set,
                   .ports = pb_ports),
    var pb_port_uuid = FlatMap(pb_ports),
    sb::Port_Binding(._uuid = pb_port_uuid, .logical_port = lsp_name),
    &SwitchPort(
        .lsp = nb.Logical_Switch_Port{._uuid = lsp_uuid, .name = lsp_name},
        .sw = switch).
```

- Aggregate the flattened IgmpSwitchGroupPort (implicitly from all ovn-controller instances) grouping by adress and logical switch:

```
relation IgmpSwitchMulticastGroup(
    address: string,
    switch : Ref<Switch>,
    ports  : Set<uuid>
)

IgmpSwitchMulticastGroup(address, switch, ports) :-
    IgmpSwitchGroupPort(address, switch, port),
    var ports = port.group_by((address, switch)).to_set().
```

At this point we have all the feature configuration relevant information stored in DDlog relations in ovn-northd-ddlog memory.

Pitfalls of projections

A projection is a join that uses only some of the data in a record. When the fields that are used have duplicates, the result can be many "copies" of a record, which DDlog represents internally with an integer "weight" that counts the number of copies. We don't have a projection with duplicates in this example, but *lswitch.dl* has many of them, such as this one:

```
relation LogicalSwitchHasACLs(ls: uuid, has_acls: bool)

LogicalSwitchHasACLs(ls, true) :-
    LogicalSwitchACL(ls, _).

LogicalSwitchHasACLs(ls, false) :-
    nb::Logical_Switch(._uuid = ls),
    not LogicalSwitchACL(ls, _).
```

When multiple projections get joined together, the weights can overflow, which causes DDlog to malfunction. The solution is to make the relation an output relation, which causes DDlog to filter it through a "distinct" operator that reduces the weights to 1. Thus, *LogicalSwitchHasACLs* is actually implemented this way:

```
output relation LogicalSwitchHasACLs(ls: uuid, has_acls: bool)
```

For more information, see Avoiding weight overflow in the DDlog tutorial.

3.6.5 Write rules to update output relations

The developer updates output tables by writing rules that generate `Out_*` relations. For IP Multicast this means:

```
/* IP_Multicast table (only applicable for Switches). */
sb::Out_IP_Multicast(._uuid = hash128(cfg.datapath),
                     .datapath = cfg.datapath,
                     .enabled = set_singleton(cfg.enabled),
                     .querier = set_singleton(cfg.querier)) :-
    &McastSwitchCfg[cfg].
```

Note: `OVN_Southbound.dl` also contains an `IP_Multicast` relation with `input` qualifier. This relation stores the current snapshot of the OVSDB table and cannot be written to.

3.6.6 Generate `Logical_Flow` and/or other forwarding records

At this point we have defined all DDlog relations required to generate `Logical_Flow``s. All we have to do is write the rules to do so. For each ``IgmpSwitchMulticastGroup` we generate a `Flow` that has as action `"outport = <Multicast_Group>; output;"`:

```
/* Ingress table 17: Add IP multicast flows learnt from IGMP (priority 90). */
for (IgmpSwitchMulticastGroup(.address = address, .switch = &sw)) {
    Flow(.logical_datapath = sw.dpname,
         .stage            = switch_stage(IN, L2_LKUP),
         .priority         = 90,
         .__match          = "eth.mcast && ip4 && ip4.dst == ${address}",
         .actions          = "outport = \"${address}\"; output;",
         .external_ids     = map_empty())
}
```

In some cases generating a logical flow is not enough. For IGMP we also need to maintain OVN southbound `Multicast_Group` records, one per IGMP group storing the corresponding `Port_Binding` uuids of ports where multicast traffic should be sent. This is also relatively straightforward:

```
/* Create a multicast group for each IGMP group learned by a Switch.
 * 'tunnel_key' == 0 triggers an ID allocation later.
 */
sb::Out_Multicast_Group (.datapath   = switch.dpname,
                         .name       = address,
                         .tunnel_key = 0,
                         .ports      = set_map_uuid2name(port_ids)) :-
    IgmpSwitchMulticastGroup(address, &switch, port_ids).
```

We must also define DDlog relations that will allocate `tunnel_key` values. There are two cases: tunnel keys for records that already existed in the database are preserved to implement stable id allocation; new multicast groups need new keys. This kind of allocation can be tricky, especially to new users of DDlog. OVN contains multiple instances of allocation, so it's probably worth reading through the existing cases and following their pattern, and, if it's still tricky, asking for assistance.

3.6.7 Appendix A. Additional relations generated by `ovsdb2ddlog`

ovsdb2ddlog generates some extra relations to manage communication with the OVSDB server. It generates records in the following relations when rows in OVSDB output tables need to be added or deleted or updated.

In the steady state, when everything is working well, a given record stays in any one of these relations only briefly: just long enough for `ovn-northd-ddlog` to send a transaction to the OVSDB server. When the OVSDB server applies the update and sends an acknowledgement, this ordinarily means that these relations become empty, because there are no longer any further changes to send.

Thus, records that persist in one of these relations is a sign of a problem. One example of such a problem is the database server rejecting the transactions sent by `ovn-northd-ddlog`, which might happen if, for example, a bug in a `.dl` file would cause some OVSDB constraint or relational integrity rule to be violated. (Such a problem can often be diagnosed by looking in the OVSDB server's log.)

- `DeltaPlus_IP_Multicast` used by the DDlog program to track new records that are not yet added to the database:

```
output relation DeltaPlus_IP_Multicast (
    datapath: uuid_or_string_t,
    enabled: Set<bool>,
    querier: Set<bool>
)
```

- `DeltaMinus_IP_Multicast` used by the DDlog program to track records that are no longer needed in the database and need to be removed:

```
output relation DeltaMinus_IP_Multicast (
    _uuid: uuid
)
```

- `Update_IP_Multicast` used by the DDlog program to track records whose fields need to be updated in the database:

```
output relation Update_IP_Multicast (
    _uuid: uuid,
    enabled: Set<bool>,
    querier: Set<bool>
)
```

CHAPTER 4

Deep Dive

How OVN is implemented and, where necessary, why it was implemented that way.

4.1 OVN

4.1.1 Debugging the DDlog version of ovn-northd

This document gives some tips for debugging correctness issues in the DDlog implementation of `ovn-northd`. To keep things conrete, we assume here that a failure occurred in one of the test cases in `ovn-e2e.at`, but the same methodology applies in any other environment. If none of these methods helps, ask for assistance or submit a bug report.

Before trying these methods, you may want to check the northd log file, `tests/testsuite.dir/<test_number>/northd/ovn-northd.log` for error messages that might explain the failure.

Compare OVSDB tables generated by DDlog vs C

The first thing I typically want to check when `ovn-northd-ddlog` does not behave as expected is how the OVSDB tables computed by DDlog differ from what the C implementation produces. Fortunately, all the infrastructure needed to do this already exists in OVN.

First, let's modify the test script, e.g., `ovn.at` to dump the contents of OVSDB right before the failure. The most common issue is a difference between the logical flows generated by the two implementations. To make it easy to compare the generated flows, make sure that the test contains something like this in the right place:

```
ovn-sbctl dump-flows > sbflows
AT_CAPTURE_FILE([sbflows])
```

The first line above dumps the OVN logical flow table to a file named `sbflows`. The second line ensures that, if the test fails, `sbflows` get logged to `testsuite.log`. That is not particularly useful for us right now, but it means that if someone later submits a bug report, that's one more piece of data that we don't have to ask for them to submit along with it.

Next, we want to run the test twice, with the C and DDlog versions of northd, e.g., `make check -j6 TESTSUITEFLAGS="-d 111 112"` if 111 and 112 are the C and DDlog versions of the same test. The `-d` in this command line makes the test driver keep test directories around even for tests that succeed, since by default it deletes them.

Now you can look at `sbflows` in each test log directory. The `ovn-northd-ddlog` developers have gone to some trouble to make the DDlog flows as similar as possible to the C ones, right down to white space and other formatting. Thus, the DDlog output is often identical to C aside from logical datapath UUIDs.

Usually, this means that one can get informative results by running `diff`, e.g.:

```
diff -u tests/testsuite.dir/111/sbflows tests/testsuite.dir/111/sbflows
```

Running the input through the `uuidfilt` utility from OVS will generally get rid of the logical datapath UUID differences as well:

```
diff -u <(uuidfilt tests/testsuite.dir/111/sbflows) <(uuidfilt tests/testsuite.dir/
↪111/sbflows)
```

If there are nontrivial differences, this often identifies your bug.

Often, once you have identified the difference between the two OVSDB dumps, this will immediately lead you to the root cause of the bug, but if you are not this lucky then the next method may help.

Record and replay DDlog execution

DDlog offers a way to record all input table updates throughout the execution of northd and replay them against DDlog running as a standalone executable without all other OVN components. This has two advantages. First, this allows one to easily tweak the inputs, e.g. to simplify the test scenario. Second, the recorded execution can be easily replayed anywhere without having to reproduce your OVN setup.

Use the `--ddlog-record` option to record updates, e.g. `--ddlog-record=replay.dat` to record to `replay.dat`. (OVN's built-in tests automatically do this.) The file contains the log of transactions in the DDlog command format (see https://github.com/vmware/differential-datalog/blob/master/doc/command_reference/command_reference.md).

To replay the log, you will need the standalone DDlog executable. By default, the build system does not compile this program, because it increases the already long Rust compilation time. To build it, add `NORTHD_CLI=1` to the `make` command line, e.g. `make NORTHD_CLI=1`.

You can modify the log before replaying it, e.g., adding `dump <table>` commands to dump the contents of relations at various points during execution. The <table> name must be fully qualified based on the file in which it is declared, e.g. `OVN_Southbound::<table>` for southbound tables or `lrouter::<table>.` for `lrouter.dl`. You can also use `dump` without an argument to dump the contents of all tables.

The following command replays the log generated by OVN test number 112 and dumps the output of DDlog to `replay.dump`:

```
northd/ovn_northd_ddlog/target/release/ovn_northd_cli < tests/testsuite.dir/112/
↪northd/replay.dat > replay.dump
```

Or, to dump just the table contents following the run, without having to edit `replay.dat`:

```
(cat tests/testsuite.dir/112/northd/replay.dat; echo 'dump;') | northd/ovn_northd_
↪ddlog/target/release/ovn_northd_cli --no-delta --no-init-snapshot > replay.dump
```

Depending on whether and how you installed OVS and OVN, you might need to point `LD_LIBRARY_PATH` to library build directories to get the CLI to run, e.g.:

```
export LD_LIBRARY_PATH=$HOME/ovn/_build/lib/.libs:$HOME/ovs/_build/lib/.libs
```

Note: The replay output may be less informative than you expect because DDlog does not, by default, keep around enough information to include input relation and intermediate relations in the output. These relations are often critical to understanding what is going on. To include them, add the options `--output-internal-relations --output-input-relations=In_` to `DDLOG_EXTRA_FLAGS` for building `ovn-northd-ddlog`. For example, `configure` as:

```
./configure DDLOG_EXTRA_FLAGS='--output-internal-relations --output-input-
↪relations=In_'
```

Debugging by Logging

One limitation of the previous method is that it allows one to inspect inputs and outputs of a rule, but not the (sometimes fairly complicated) computation that goes on inside the rule. You can of course break up the rule into several rules and dump the intermediate outputs.

There are at least two alternatives for generating log messages. First, you can write rules to add strings to the Warning relation declared in `ovn_north.dl`. Code in `ovn-northd-ddlog.c` will log any given string in this relation just once, when it is first added to the relation. (If it is removed from the relation and then added back later, it will be logged again.)

Second, you can call using the `warn()` function declared in `ovn.dl` from a DDlog rule. It's not straightforward to know exactly when this function will be called, like it would be in an imperative language like C, since DDlog is a declarative language where the user doesn't directly control when rules are triggered. You might, for example, see the rule being triggered multiple times with the same input. Nevertheless, this debugging technique is useful in practice.

You will find many examples of the use of Warning and `warn` in `ovn_northd.dl`, where it is frequently used to report non-critical errors.

Debugging panics

TODO: update these instructions as DDlog's internal handling of panic's is improved.

DDlog is a safe language, so DDlog programs normally do not crash, except for the following three cases:

- A panic in a Rust function imported to DDlog as `extern function`.
- A panic in a C function imported to DDlog as `extern function`.
- A bug in the DDlog runtime or libraries.

Below we walk through the steps involved in debugging such failures. In this scenario, there is an array-index-out-of-bounds error in the `ovn_scan_static_dynamic_ip6()` function, which is written in Rust and imported to DDlog as an `extern function`. When invoked from a DDlog rule, this function causes a panic in one of DDlog worker threads.

Step 1: Check for error messages in the northd log. A panic can generally lead to unpredictable outcomes, so one cannot count on a clean error message showing up in the log (Other outcomes include crashing the entire process and even deadlocks. We are working to eliminate the latter possibility). In this case we are lucky to observe a bunch of error messages like the following in the `northd` log:

```
2019-09-23T16:23:24.549Z|00011|ovn_northd|ERR|ddlog_transaction_commit():
error:  failed to receive flush ack message from timely dataflow
thread
```

These messages are telling us that something is broken inside the DDlog runtime.

Step 2: Record and replay the failing scenario. We use DDlog's record/replay capabilities (see above) to capture the faulty scenario. We replay the recorded trace:

```
northd/ovn_northd_ddlog/target/release/ovn_northd_cli < tests/testsuite.dir/117/
↪northd/replay.dat
```

This generates a bunch of output ending with:

```
thread 'worker thread 2' panicked at 'index out of bounds: the len is 1 but the index␣
↪is 1', /rustc/eae3437dfe991621e8afdc82734f4a172d7ddf9b/src/libcore/slice/mod.
↪rs:2681:10
note: run with RUST_BACKTRACE=1 environment variable to display a backtrace.
```

We re-run the CLI again with backtrace enabled (as suggested by the error message):

```
RUST_BACKTRACE=1 northd/ovn_northd_ddlog/target/release/ovn_northd_cli < tests/
↪testsuite.dir/117/northd/replay.dat
```

This finally yields the following stack trace, which suggests array bound violation in `ovn_scan_static_dynamic_ip6`:

```
0: backtrace::backtrace::libunwind::trace
          at /cargo/registry/src/github.com-1ecc6299db9ec823/backtrace-0.3.29  10:␣
↪core::panicking::panic_bounds_check
          at src/libcore/panicking.rs:61
[SKIPPED]
11: ovn_northd_ddlog::__ovn::ovn_scan_static_dynamic_ip6
12: ovn_northd_ddlog::prog::__f
[SKIPPED]
```

Finally, looking at the source code of `ovn_scan_static_dynamic_ip6`, we identify the following line, containing an unsafe array indexing operator, as the culprit:

```
ovn_ipv6_parse(&f[1].to_string())
```

Clean build

Occasionally it's desirable to a full and complete build of the DDlog-generated code. To trigger that, delete the generated `ovn_northd_ddlog` directory and the `ddlog.stamp` witness file, like this:

```
rm -rf northd/ovn_northd_ddlog northd/ddlog.stamp
```

or:

```
make clean-ddlog
```

Submitting a bug report

If you are having trouble with DDlog and the above methods do not help, please submit a bug report to `bugs@openvswitch.org`, CC `ryzhyk@gmail.com`.

In addition to problem description, please provide as many of the following as possible:

- Are you running with the right DDlog for the version of OVN? OVN and DDlog are both evolving and OVN needs to build against a specific version of DDlog.
- `replay.dat` file generated as described above
- Logs: `ovn-northd.log` and `testsuite.log`, if you are running the OVN test suite

4.1.2 Integration Guide for Centralized Control

HA for OVN DB servers using pacemaker

The ovsdb servers can work in either active or backup mode. In backup mode, db server will be connected to an active server and replicate the active servers contents. At all times, the data can be transacted only from the active server. When the active server dies for some reason, entire OVN operations will be stalled.

Pacemaker is a cluster resource manager which can manage a defined set of resource across a set of clustered nodes. Pacemaker manages the resource with the help of the resource agents. One among the resource agent is OCF

OCF is nothing but a shell script which accepts a set of actions and returns an appropriate status code.

With the help of the OCF resource agent ovn/utilities/ovndb-servers.ocf, one can defined a resource for the pacemaker such that pacemaker will always maintain one running active server at any time.

After creating a pacemaker cluster, use the following commands to create one active and multiple backup servers for OVN databases:

```
$ pcs resource create ovndb_servers ocf:ovn:ovndb-servers \
    master_ip=x.x.x.x \
    ovn_ctl=<path of the ovn-ctl script> \
    op monitor interval="10s" \
    op monitor role=Master interval="15s"
$ pcs resource master ovndb_servers-master ovndb_servers \
   meta notify="true"
```

The *master_ip* and *ovn_ctl* are the parameters that will be used by the OCF script. *ovn_ctl* is optional, if not given, it assumes a default value of /usr/share/openvswitch/scripts/ovn-ctl. *master_ip* is the IP address on which the active database server is expected to be listening, the slave node uses it to connect to the master node. You can add the optional parameters 'nb_master_port', 'nb_master_protocol', 'sb_master_port', 'sb_master_protocol' to set the protocol and port.

Whenever the active server dies, pacemaker is responsible to promote one of the backup servers to be active. Both ovn-controller and ovn-northd needs the ip-address at which the active server is listening. With pacemaker changing the node at which the active server is run, it is not efficient to instruct all the ovn-controllers and the ovn-northd to listen to the latest active server's ip-address.

This problem can be solved by two ways:

1. By using a native ocf resource agent `ocf:heartbeat:IPaddr2`. The IPAddr2 resource agent is just a resource with an ip-address. When we colocate this resource with the active server, pacemaker will enable the active server to be connected with a single ip-address all the time. This is the ip-address that needs to be given as the parameter while creating the *ovndb_servers* resource.

Use the following command to create the IPAddr2 resource and colocate it with the active server:

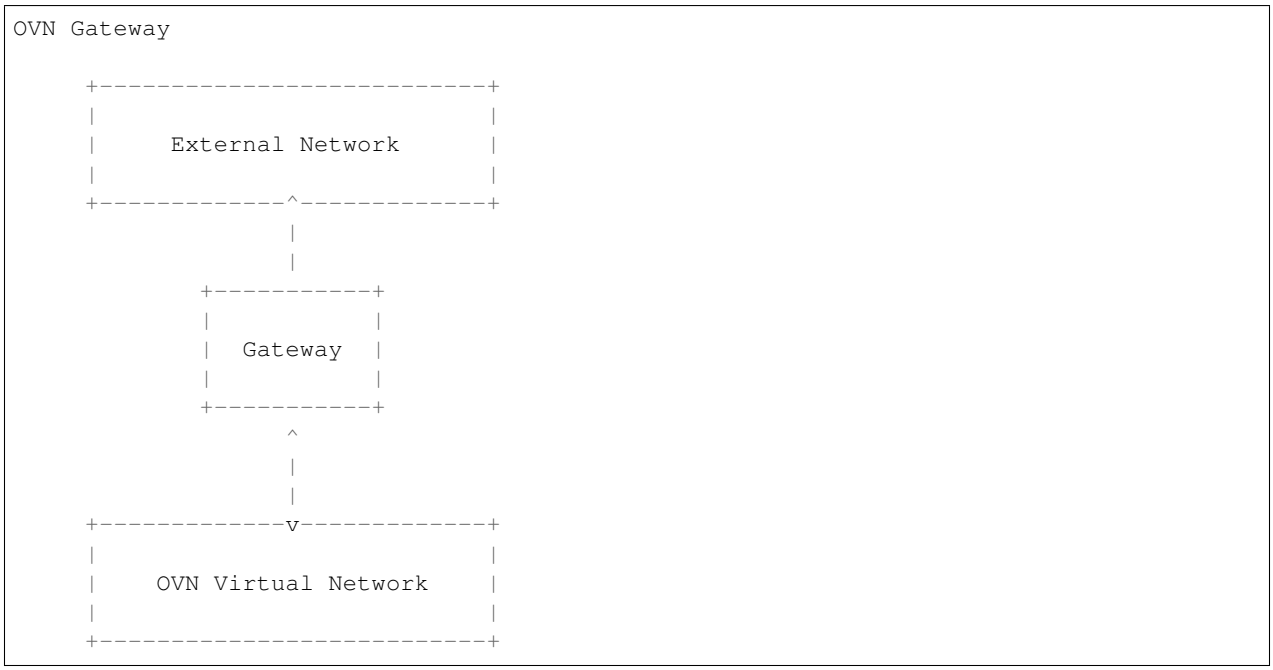
```
$ pcs resource create VirtualIP ocf:heartbeat:IPaddr2 ip=x.x.x.x \
   op monitor interval=30s
$ pcs constraint order promote ovndb_servers-master then VirtualIP
$ pcs constraint colocation add VirtualIP with master ovndb_servers-master \
   score=INFINITY
```

2. Using load balancer vip ip as a master_ip. In order to use this feature, one needs to use listen_on_master_ip_only to no. Current code for load balancer have been tested to work with tcp protocol and needs to be tested/enchanced for ssl. Using load balancer, standby nodes will not listen on nb and sb db ports so that load balancer will always communicate to the active node and all the traffic will be sent to active node only. Standby will continue to sync using LB VIP IP in this case.

Use the following command to create pcs resource using LB VIP IP:

```
$ pcs resource create ovndb_servers ocf:ovn:ovndb-servers \
     master_ip="<load_balance_vip_ip>" \
     listen_on_master_ip_only="no" \
     ovn_ctl=<path of the ovn-ctl script> \
     op monitor interval="10s" \
     op monitor role=Master interval="15s"
$ pcs resource master ovndb_servers-master ovndb_servers \
    meta notify="true"
```

4.1.3 OVN Gateway High Availability Plan



The OVN gateway is responsible for shuffling traffic between the tunneled overlay network (governed by ovn-northd), and the legacy physical network. In a naive implementation, the gateway is a single x86 server, or hardware VTEP. For most deployments, a single system has enough forwarding capacity to service the entire virtualized network, however, it introduces a single point of failure. If this system dies, the entire OVN deployment becomes unavailable. To mitigate this risk, an HA solution is critical – by spreading responsibility across multiple systems, no single server failure can take down the network.

An HA solution is both critical to the manageability of the system, and extremely difficult to get right. The purpose of this document, is to propose a plan for OVN Gateway High Availability which takes into account our past experience building similar systems. It should be considered a fluid changing proposal, not a set-in-stone decree.

Note: This document describes a range of options OVN could take to provide high availability for gateways. The current implementation provides L3 gateway high availability by the "Router Specific Active/Backup" approach de-

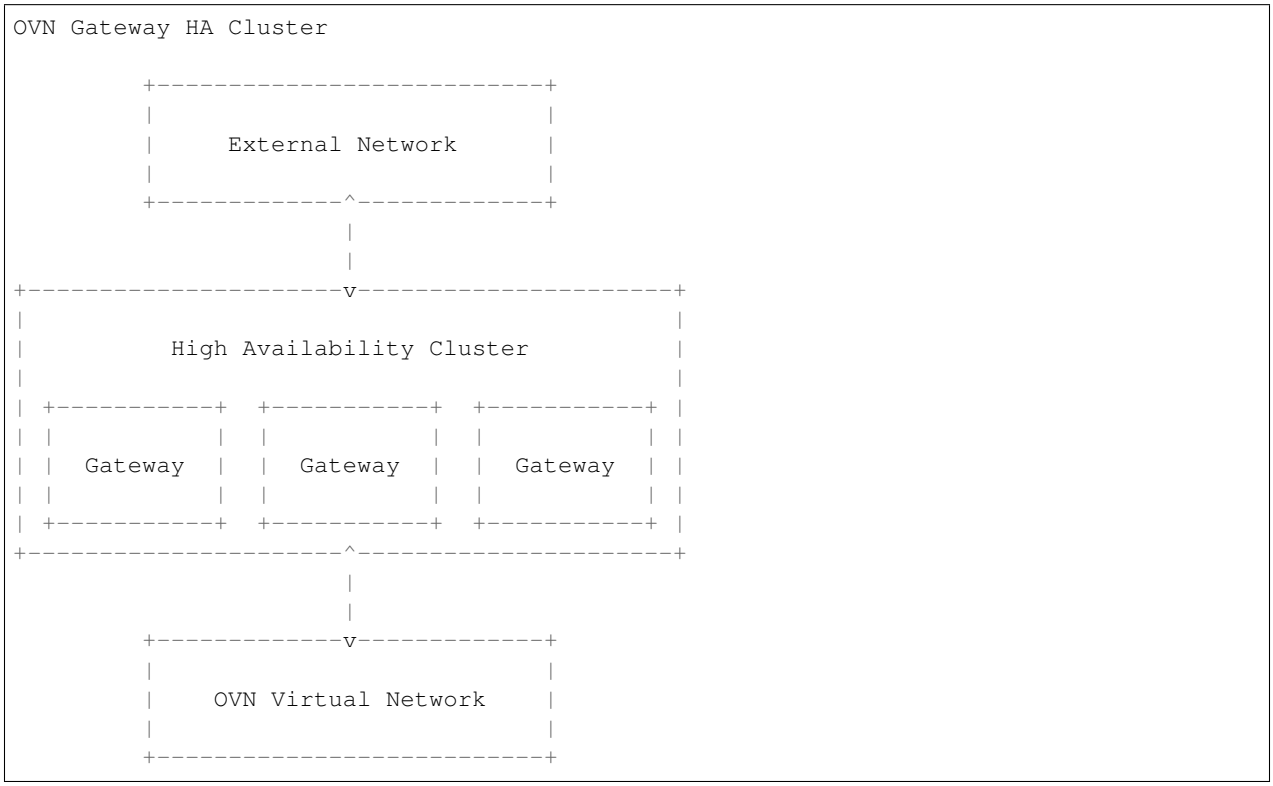
scribed in this document.

Basic Architecture

In an OVN deployment, the set of hypervisors and network elements operating under the guidance of ovn-northd are in what's called "logical space". These servers use VXLAN, STT, or Geneve to communicate, oblivious to the details of the underlying physical network. When these systems need to communicate with legacy networks, traffic must be routed through a Gateway which translates from OVN controlled tunnel traffic, to raw physical network traffic.

Since the gateway is typically the only system with a connection to the physical network all traffic between logical space and the WAN must travel through it. This makes it a critical single point of failure – if the gateway dies, communication with the WAN ceases for all systems in logical space.

To mitigate this risk, multiple gateways should be run in a "High Availability Cluster" or "HA Cluster". The HA cluster will be responsible for performing the duties of a gateways, while being able to recover gracefully from individual member failures.



L2 vs L3 High Availability

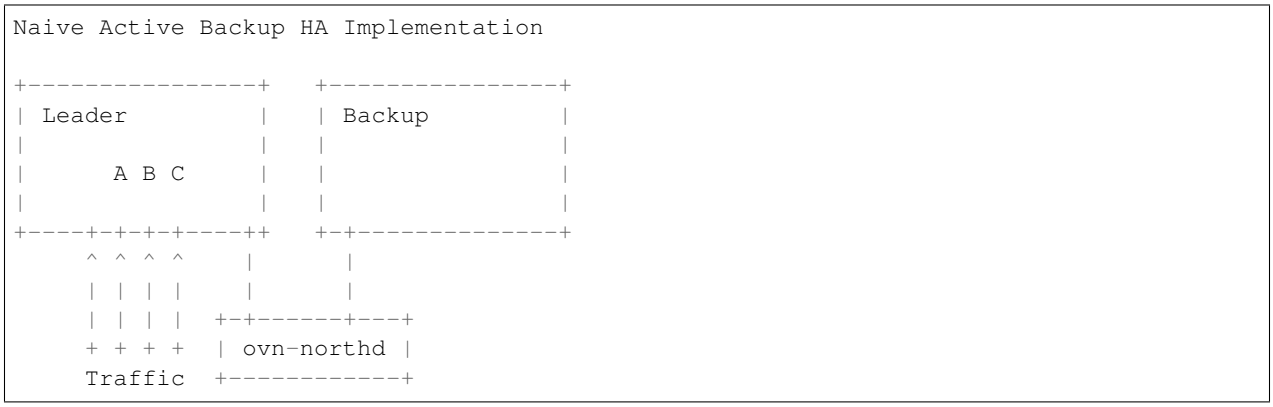
In order to achieve this goal, there are two broad approaches one can take. The HA cluster can appear to the network like a giant Layer 2 Ethernet Switch, or like a giant IP Router. These approaches are called L2HA, and L3HA respectively. L2HA allows ethernet broadcast domains to extend into logical space, a significant advantage, but this comes at a cost. The need to avoid transient L2 loops during failover significantly complicates their design. On the other hand, L3HA works for most use cases, is simpler, and fails more gracefully. For these reasons, it is suggested that OVN supports an L3HA model, leaving L2HA for future work (or third party VTEP providers). Both models are discussed further below.

L3HA

In this section, we'll work through a basic simple L3HA implementation, on top of which we'll gradually build more sophisticated features explaining their motivations and implementations as we go.

Naive active-backup

Let's assume that there are a collection of logical routers which a tenant has asked for, our task is to schedule these logical routers on one of N gateways, and gracefully redistribute the routers on gateways which have failed. The absolute simplest way to achieve this is what we'll call "naive-active-backup".



In a naive active-backup, one of the Gateways is chosen (arbitrarily) as a leader. All logical routers (A, B, C in the figure), are scheduled on this leader gateway and all traffic flows through it. ovn-northd monitors this gateway via OpenFlow echo requests (or some equivalent), and if the gateway dies, it recreates the routers on one of the backups.

This approach basically works in most cases and should likely be the starting point for OVN – it's strictly better than no HA solution and is a good foundation for more sophisticated solutions. That said, it's not without it's limitations. Specifically, this approach doesn't coordinate with the physical network to minimize disruption during failures, and it tightly couples failover to ovn-northd (we'll discuss why this is bad in a bit), and wastes resources by leaving backup gateways completely unutilized.

Router Failover

When ovn-northd notices the leader has died and decides to migrate routers to a backup gateway, the physical network has to be notified to direct traffic to the new gateway. Otherwise, traffic could be blackholed for longer than necessary making failovers worse than they need to be.

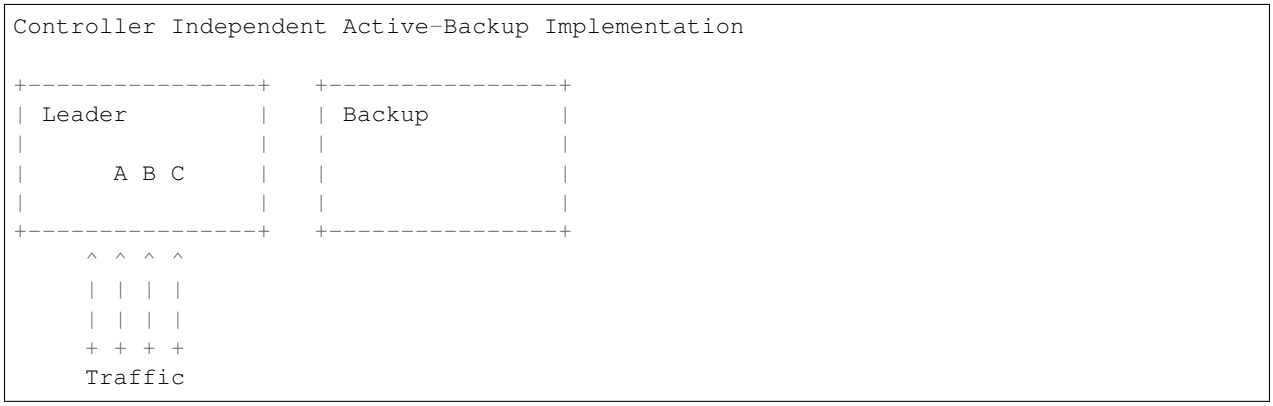
For now, let's assume that OVN requires all gateways to be on the same IP subnet on the physical network. If this isn't the case, gateways would need to participate in routing protocols to orchestrate failovers, something which is difficult and out of scope of this document.

Since all gateways are on the same IP subnet, we simply need to worry about updating the MAC learning tables of the Ethernet switches on that subnet. Presumably, they all have entries for each logical router pointing to the old leader. If these entries aren't updated, all traffic will be sent to the (now defunct) old leader, instead of the new one.

In order to mitigate this issue, it's recommended that the new gateway sends a Reverse ARP (RARP) onto the physical network for each logical router it now controls. A Reverse ARP is a benign protocol used by many hypervisors when virtual machines migrate to update L2 forwarding tables. In this case, the ethernet source address of the RARP is that of the logical router it corresponds to, and its destination is the broadcast address. This causes the RARP to travel to every L2 switch in the broadcast domain, updating forwarding tables accordingly. This strategy is recommended in

all failover mechanisms discussed in this document – when a router newly boots on a new leader, it should RARP its MAC address.

Controller Independent Active-backup



The fundamental problem with naive active-backup, is it tightly couples the failover solution to ovn-northd. This can significantly increase downtime in the event of a failover as the (often already busy) ovn-northd controller has to recompute state for the new leader. Worse, if ovn-northd goes down, we can't perform gateway failover at all. This violates the principle that control plane outages should have no impact on dataplane functionality.

In a controller independent active-backup configuration, ovn-northd is responsible for initial configuration while the HA cluster is responsible for monitoring the leader, and failing over to a backup if necessary. ovn-northd sets HA policy, but doesn't actively participate when failovers occur.

Of course, in this model, ovn-northd is not without some responsibility. Its role is to pre-plan what should happen in the event of a failure, leaving it to the individual switches to execute this plan. It does this by assigning each gateway a unique leadership priority. Once assigned, it communicates this priority to each node it controls. Nodes use the leadership priority to determine which gateway in the cluster is the active leader by using a simple metric: the leader is the gateway that is healthy, with the highest priority. If that gateway goes down, leadership falls to the next highest priority, and conversely, if a new gateway comes up with a higher priority, it takes over leadership.

Thus, in this model, leadership of the HA cluster is determined simply by the status of its members. Therefore if we can communicate the status of each gateway to each transport node, they can individually figure out which is the leader, and direct traffic accordingly.

Tunnel Monitoring

Since in this model leadership is determined exclusively by the health status of member gateways, a key problem is how do we communicate this information to the relevant transport nodes. Luckily, we can do this fairly cheaply using tunnel monitoring protocols like BFD.

The basic idea is pretty straightforward. Each transport node maintains a tunnel to every gateway in the HA cluster (not just the leader). These tunnels are monitored using the BFD protocol to see which are alive. Given this information, hypervisors can trivially compute the highest priority live gateway, and thus the leader.

In practice, this leadership computation can be performed trivially using the bundle or group action. Rather than using OpenFlow to simply output to the leader, all gateways could be listed in an active-backup bundle action ordered by their priority. The bundle action will automatically take into account the tunnel monitoring status to output the packet to the highest priority live gateway.

Inter-Gateway Monitoring

One somewhat subtle aspect of this model, is that failovers are not globally atomic. When a failover occurs, it will take some time for all hypervisors to notice and adjust accordingly. Similarly, if a new high priority Gateway comes up, it may take some time for all hypervisors to switch over to the new leader. In order to avoid confusing the physical network, under these circumstances it's important for the backup gateways to drop traffic they've received erroneously. In order to do this, each Gateway must know whether or not it is, in fact active. This can be achieved by creating a mesh of tunnels between gateways. Each gateway monitors the other gateways its cluster to determine which are alive, and therefore whether or not that gateway happens to be the leader. If leading, the gateway forwards traffic normally, otherwise it drops all traffic.

We should note that this method works well under the assumption that there are no inter-gateway connectivity failures, in such case this method would fail to elect a single master. The simplest example is two gateways which stop seeing each other but can still reach the hypervisors. Protocols like VRRP or CARP have the same issue. A mitigation for this type of failure mode could be achieved by having all network elements (hypervisors and gateways) periodically share their link status to other endpoints.

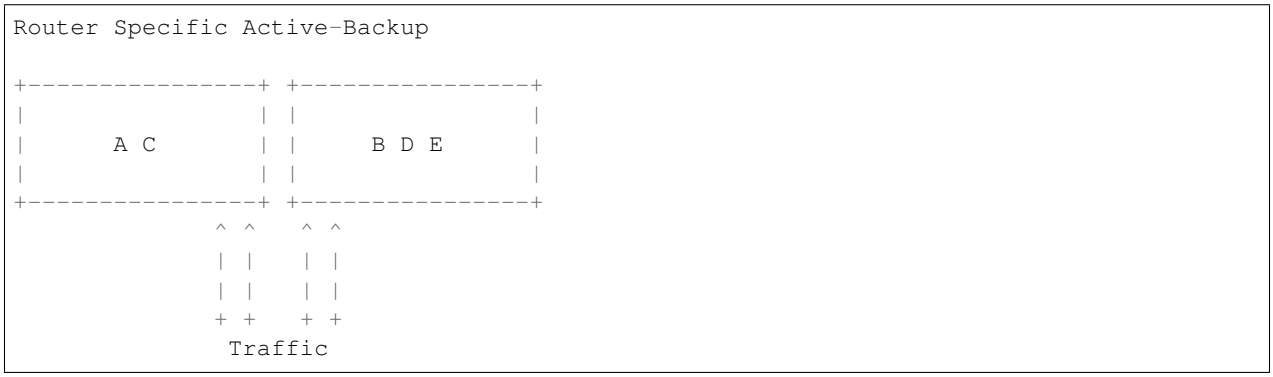
Gateway Leadership Resignation

Sometimes a gateway may be healthy, but still may not be suitable to lead the HA cluster. This could happen for several reasons including:

- The physical network is unreachable
- BFD (or ping) has detected the next hop router is unreachable
- The Gateway recently booted and isn't fully configured

In this case, the Gateway should resign leadership by holding its tunnels down using the `other_config:cpath_down` flag. This indicates to participating hypervisors and Gateways that this gateway should be treated as if it's down, even though its tunnels are still healthy.

Router Specific Active-Backup



Controller independent active-backup is a great advance over naive active-backup, but it still has one glaring problem – it under-utilizes the backup gateways. In ideal scenario, all traffic would split evenly among the live set of gateways. Getting all the way there is somewhat tricky, but as a step in the direction, one could use the "Router Specific Active-Backup" algorithm. This algorithm looks a lot like active-backup on a per logical router basis, with one twist. It chooses a different active Gateway for each logical router. Thus, in situations where there are several logical routers, all with somewhat balanced load, this algorithm performs better.

Implementation of this strategy is quite straightforward if built on top of basic controller independent active-backup. On a per logical router basis, the algorithm is the same, leadership is determined by the liveness of the gateways. The key difference here is that the gateways must have a different leadership priority for each logical router. These leadership priorities can be computed by ovn-northd just as they had been in the controller independent active-backup model.

Once we have these per logical router priorities, they simply need be communicated to the members of the gateway cluster and the hypervisors. The hypervisors in particular, need simply have an active-backup bundle action (or group action) per logical router listing the gateways in priority order for *that router*, rather than having a single bundle action shared for all the routers.

Additionally, the gateways need to be updated to take into account individual router priorities. Specifically, each gateway should drop traffic of backup routers it's running, and forward traffic of active gateways, instead of simply dropping or forwarding everything. This should likely be done by having ovn-controller recompute OpenFlow for the gateway, though other options exist.

The final complication is that ovn-northd's logic must be updated to choose these per logical router leadership priorities in a more sophisticated manner. It doesn't matter much exactly what algorithm it chooses to do this, beyond that it should provide good balancing in the common case. I.E. each logical routers priorities should be different enough that routers balance to different gateways even when failures occur.

Preemption

In an active-backup setup, one issue that users will run into is that of gateway leader preemption. If a new Gateway is added to a cluster, or for some reason an existing gateway is rebooted, we could end up in a situation where the newly activated gateway has higher priority than any other in the HA cluster. In this case, as soon as that gateway appears, it will preempt leadership from the currently active leader causing an unnecessary failover. Since failover can be quite expensive, this preemption may be undesirable.

The controller can optionally avoid preemption by cleverly tweaking the leadership priorities. For each router, new gateways should be assigned priorities that put them second in line or later when they eventually come up. Furthermore, if a gateway goes down for a significant period of time, its old leadership priorities should be revoked and new ones should be assigned as if it's a brand new gateway. Note that this should only happen if a gateway has been down for a while (several minutes), otherwise a flapping gateway could have wide ranging, unpredictable, consequences.

Note that preemption avoidance should be optional depending on the deployment. One necessarily sacrifices optimal load balancing to satisfy these requirements as new gateways will get no traffic on boot. Thus, this feature represents a trade-off which must be made on a per installation basis.

Fully Active-Active HA

The final step in L3HA is to have true active-active HA. In this scenario each router has an instance on each Gateway, and a mechanism similar to ECMP is used to distribute traffic evenly among all instances. This mechanism would require Gateways to participate in routing protocols with the physical network to attract traffic and alert of failures. It is out of scope of this document, but may eventually be necessary.

L2HA

L2HA is very difficult to get right. Unlike L3HA, where the consequences of problems are minor, in L2HA if two gateways are both transiently active, an L2 loop triggers and a broadcast storm results. In practice to get around this, gateways end up implementing an overly conservative "when in doubt drop all traffic" policy, or they implement something like MLAG.

MLAG has multiple gateways work together to pretend to be a single L2 switch with a large LACP bond. In principle, it's the right solution to the problem as it solves the broadcast storm problem, and has been deployed successfully in other contexts. That said, it's difficult to get right and not recommended.

4.1.4 Role Based Access Control

Where SSL provides authentication when connecting to an OVS database, role based access control (RBAC) provides authorization to operations performed by clients connecting to an OVS database. RBAC allows for administrators to restrict the database operations a client may perform and thus enhance the security already provided by SSL.

In theory, any OVS database could define RBAC roles and permissions, but at present only the OVN southbound database has the appropriate tables defined to facilitate RBAC.

Mechanics

RBAC is intended to supplement SSL. In order to enable RBAC, the connection to the database must use SSL. Some permissions in RBAC are granted based on the certificate common name (CN) of the connecting client.

RBAC is controlled with two database tables, RBAC_Role and RBAC_Permission. The RBAC_Permission table contains records that describe a set of permissions for a given table in the database.

The RBAC_Permission table contains the following columns:

table The table in the database for which permissions are being described.

insert_delete Describes whether insertion and deletion of records is allowed.

update A list of columns that are allowed to be updated.

authorization A list of column names. One of the listed columns must match the SSL certificate CN in order for the attempted operation on the table to succeed. If a key-value pair is provided, then the key is the column name, and the value is the name of a key in that column. An empty string gives permission to all clients to perform operations.

The RBAC_Role table contains the following columns:

name The name of the role being defined

permissions A list of key-value pairs. The key is the name of a table in the database, and the value is a UUID of a record in the RBAC_Permission table that describes the permissions the role has for that table.

Note: All tables not explicitly referenced in an RBAC_Role record are read-only

---

In order to enable RBAC, specify the role name as an argument to the set-connection command for the database. As an example, to enable the "ovn-controller" role on the OVN southbound database, use the following command:

```
$ ovn-sbctl set-connection role=ovn-controller ssl:192.168.0.1:6642
```

Note: There is currently no pre-defined role for ovn-northd. You must configure a separate listener on the OVN southbound database that ovn-northd can connect to if your deployment topology require ovn-northd to connect to a OVN southbound database instance on a remote machine.

Pre-defined Roles

This section describes roles that have been defined internally by OVS/OVN.

ovn-controller

The ovn-controller role is specified in the OVN southbound database and is intended for use by hypervisors running the ovn-controller daemon. ovn-controller connects to the OVN southbound database mostly to read information, but there are a few cases where ovn-controller also needs to write. The ovn-controller role was designed to allow for ovn-controllers to write to the southbound database only in places where it makes sense to do so. This way, if an intruder were to take over a hypervisor running ovn-controller, it is more difficult to compromise the entire overlay network.

It is strongly recommended to set the ovn-controller role for the OVN southbound database to enhance security.

4.1.5 What's New with OVS and OVN 2.8

This document is about what was added in Open vSwitch 2.8, which was released at the end of August 2017, concentrating on the new features in OVN. It also covers some of what is coming up in Open vSwitch and OVN 2.9, which is due to be released in February 2018. OVN has many features, and this document does not cover every new or enhanced feature (but contributions are welcome).

This document assumes a basic familiarity with Open vSwitch, OVN, and their associated tools. For more information, please refer to the Open vSwitch and OVN documentation, such as the `ovn-architecture`(7) manpage.

Debugging and Troubleshooting

Before version 2.8, Open vSwitch command-line tools were far more painful to use than they needed to be. This section covers the improvements made to the CLI in the 2.8 release.

User-Hostile UUIDs

The OVN CLI, through `ovn-nbctl`, `ovn-nbctl`, and `ovn-trace`, used full-length UUIDs almost everywhere. It didn't even provide any assistance with completion, etc., which in practice meant always cutting and pasting UUIDs from one command or window to another. This problem wasn't limited to the places where one would expect to have to see or use a UUID, either. In many places where one would expect to be able to use a network, router, or port name, a UUID was required instead. In many places where one would want to see a name, the UUID was displayed instead. More than anything else, these shortcomings made the CLI user-hostile.

There was an underlying problem that the southbound database didn't actually contain all the information needed to provide a decent user interface. In some cases, for example, the human-friendly names that one would want to use for entities simply weren't part of the database. These names weren't necessary for correctness, only for usability.

OVN 2.8 eased many of these problems. Most parts of the CLI now allow the user to abbreviate UUIDs, as long as the abbreviations are unique within the database. Some parts of the CLI where full-length UUIDs make output hard to read now abbreviate them themselves. Perhaps more importantly, in many places the OVN CLI now displays and accepts human-friendly names for networks, routers, ports, and other entities. In the places where the names were not previously available, OVN (through `ovn-northd`) now copies the names into the southbound database.

The CLIs for layers below OVN, at the OpenFlow and datapath layers with `ovs-ofctl` and `ovs-dpctl`, respectively, had some similar problems in which numbers were used for entities that had human-friendly names. Open vSwitch 2.8 also solves some of those problems. Other than that, the most notable enhancement in this area was the `--no-stats` option to `ovs-ofctl dump-flows`, which made that command's output more readable for the cases where per-flow statistics were not interesting to the reader.

Connections Between Levels

OVN and Open vSwitch work almost like a stack of compilers: the OVN Neutron plugin translates Neutron configuration into OVN northbound configuration, which `ovn-northd` translates into logical flows, which `ovn-controller` translates into OpenFlow flows, which `ovs-vswitchd` translates into datapath flows. For debugging and troubleshooting it is often necessary to understand exactly how these translations work. The relationship from a logical flow to its OpenFlow flows, or in the other direction, from an OpenFlow flow back to the logical flow that produced it, was often of particular interest, but OVN didn't provide good tools for the job.

OVN 2.8 added some new features that ease these jobs. `ovn-sbctl lflow-list` has a new option `--ovs` that lists the OpenFlow flows on a particular chassis that were generated from the logical flows that it lists. `ovn-trace` also added a similar `--ovs` option that applies to the logical flows it traces.

In the other direction, OVN 2.8 added a new utility `ovn-detrace` that, given an Open vSwitch trace of OpenFlow flows, annotates it with the logical flows that yielded those OpenFlow flows.

Distributed Firewall

OVN supports a distributed firewall with stateful connection tracking to ensure that only packets for established connections, or those that the configuration explicitly allows, can ingress a given VM or container. Neutron uses this feature by default. Most packets in an OpenStack environment pass through it twice, once after egress from the packet's source VM and once before ingress into its destination VM. Before OVN 2.8, the `ovn-trace` program, which shows the path of a packet through an OVN logical network, did not support the logical firewall, which in practice made it almost useless for Neutron.

In OVN 2.8, `ovn-trace` adds support for the logical firewall. By default it assumes that packets are part of an established connection, which is usually what the user wants as part of the trace. It also accepts command-line options to override that assumption, which allows the user to discover the treatment of packets that the firewall should drop.

At the next level deeper, prior to Open vSwitch 2.8, the OpenFlow tracing command `ofproto/trace` also supported neither the connection tracking feature underlying the OVN distributed firewall nor the "recirculation" feature that accompanied it. This meant that, even if the user tried to look deeper into the distributed firewall mechanism, he or she would encounter a further roadblock. Open vSwitch 2.8 added support for both of these features as well.

Summary Display

`ovn-nbctl show` and `ovn-sbctl show`, for showing an overview of the OVN configuration, didn't show a lot of important information. OVN adds some more useful information here.

DNS, and IPAM

OVN 2.8 adds a built-in DNS server designed for assigning names to VMs and containers within an OVN logical network. DNS names are assigned using records in the OVN northbound database and, like other OVN features, translated into logical flows at the OVN southbound layer. DNS requests directed to the OVN DNS server never leave the hypervisor from which the request is sent; instead, OVN processes and replies to the request from its `ovn-controller` local agent. The OVN DNS server is not a general-purpose DNS server and cannot be used for that purpose.

OVN includes simple built-in support for IP address management (IPAM), in which OVN assigns IP addresses to VMs or containers from a pool or pools of IP addresses delegated to it by the administrator. Before OVN 2.8, OVN IPAM only supported IPv4 addresses; OVN 2.8 adds support for IPv6. OVN 2.8 also enhances the address pool support to allow specific addresses to be excluded. Neutron assigns IP addresses itself and does not use OVN IPAM.

High Availability

As a distributed system, in OVN a lot can go wrong. As OVN advances, it adds redundancy in places where currently a single failure could disrupt the functioning of the system as a whole. OVN 2.8 adds two new kinds of high availability.

ovn-northd HA

The `ovn-northd` program sits between the OVN northbound and southbound databases and translates from a logical network configuration into logical flows. If `ovn-northd` itself or the host on which it runs fails, then updates to the OVN northbound configuration will not propagate to the hypervisors and the OVN configuration freezes in place until `ovn-northd` restarts.

OVN 2.8 adds support for active-backup HA to `ovn-northd`. When more than one `ovn-northd` instance runs, it uses an OVSDB locking feature to automatically choose a single active instance. When that instance dies or becomes nonresponsive, the OVSDB server automatically choose one of the remaining instance(s) to take over.

L3 Gateway HA

In OVN 2.8, multiple chassis may now be specified for L3 gateways. When more than one chassis is specified, OVN manages high availability for that gateway. Each hypervisor uses the BFD protocol to keep track of the gateway nodes that are currently up. At any given time, a hypervisor uses the highest-priority gateway node that is currently up.

OVSDB

The OVN architecture relies heavily on OVSDB, the Open vSwitch database, for hosting the northbound and southbound databases. OVSDB was originally selected for this purpose because it was already used in Open vSwitch for configuring OVS itself and, thus, it was well integrated with OVS and well supported in C and Python, the two languages that are used in Open vSwitch.

OVSDB was well designed for its original purpose of configuring Open vSwitch. It supports ACID transactions, has a small, efficient server, a flexible schema system, and good support for troubleshooting and debugging. However, it lacked several features that are important for OVN but not for Open vSwitch. As OVN advances, these missing features have become more and more of a problem. One option would be to switch to a different database that already has many of these features, but despite a careful search, no ideal existing database was identified, so the project chose instead to improve OVSDB where necessary to bring it up to speed. The following sections talk more about recent and future improvements.

High Availability

When `ovsdb-server` was only used for OVS configuration, high availability was not important. `ovsdb-server` was capable of restarting itself automatically if it crashed, and if the whole system went down then Open vSwitch itself was dead too, so the database server's failure was not important.

In contrast, the northbound and southbound databases are centralized components of a distributed system, so it is important that they not be a single point of failure for the system as a whole. In released versions of OVN, `ovsdb-server` supports only "active-backup replication" across a pair of servers. This means that if one server goes down, the other can pick it back up approximately where the other one left off. The servers do not have built-in support for deciding at any given time which is the active and which the backup, so the administrator must configure an external agent to do this management.

Active-backup replication is not entirely satisfactory, for multiple reasons. Replication is only approximate. Configuring the external agent requires extra work. There is no benefit from the backup server except when the active server fails. At most two servers can be used.

A new form of high availability for OVSDB is under development for the OVN 2.9 release, based on the Raft algorithm for distributed consensus. Whereas replication uses two servers, clustering using Raft requires three or more (typically an odd number) and continues functioning as long as more than half of the servers are up. The clustering implementation is built into `ovsdb-server` and does not require an external agent. Clustering preserves the ACID properties of the database, so that a transaction that commits is guaranteed to persist. Finally, reads (which are the bulk of the OVN workload) scale with the size of the cluster, so that adding more servers should improve performance as the number of hypervisors in an OVN deployment increases. As of this writing, OVSDB support for clustering is undergoing development and early deployment testing.

RBAC security

Until Open vSwitch 2.8, `ovsdb-server` had little support for access control within a database. If an OVSDB client could modify the database at all, it could make arbitrary changes. This was sufficient for most uses case to that point.

Hypervisors in an OVN deployment need access to the OVN southbound database. Most of their access is reads, to find out about the OVN configuration. Hypervisors do need some write access to the southbound database, primarily to let the other hypervisors know what VMs and containers they are running and how to reach them. Thus, OVN gives all of the hypervisors in the OVN deployment write access to the OVN southbound database. This is fine when all is well, but if any of the hypervisors were compromised then they could disrupt the entire OVN deployment by corrupting the database.

The OVN developers considered a few ways to solve this problem. One way would be to introduce a new central service (perhaps in `ovn-northd`) that provided only the kinds of writes that the hypervisors legitimately need, and then grant hypervisors direct access to the southbound database only for reads. But ultimately the developers decided to introduce a new form of more access control for OVSDB, called the OVSDB RBAC (role-based access control) feature. OVSDB RBAC allows for granular enough control over access that hypervisors can be granted only the ability to add, modify, and delete the records that relate to themselves, preventing them from corrupting the database as a whole.

Further Directions

For more information about new features in OVN and Open vSwitch, please refer to the NEWS file distributed with the source tree. If you have questions about Open vSwitch or OVN features, please feel free to write to the Open vSwitch discussion mailing list at ovs-discuss@openvswitch.org.

4.1.6 VIF Plug Providers

VIF Plug Providers

Traditionally it has been the CMSes responsibility to create VIFs as part of instance life cycle, and subsequently manage plug/unplug operations on the integration bridge following the conventions described in the Open vSwitch Integration Guide for mapping of VIFs to OVN logical port.

With the advent of NICs connected to multiple distinct CPUs we can have a topology where the instance runs on one host and Open vSwitch and OVN runs on a different host, the smartnic control plane CPU. The host facing interfaces will be visible to Open vSwitch and OVN as representor ports.

The actions necessary for plugging and unplugging the representor port in Open vSwitch running on the smartnic control plane CPU would be the same for every CMS.

Instead of every CMS having to develop their own version of an agent to do the plugging, we provide a pluggable infrastructure in OVN that allows the *ovn-controller* to perform the plugging on CMS direction.

Hardware or platform specific details for initialization and lookup of representor ports is provided by an plugging provider library hosted inside or outside the core OVN repository, and linked at OVN build time.

Life Cycle of an OVN plugged VIF

1. CMS creates a record in the OVN Northbound Logical_Switch_Port table with the options column containing the *vif-plug-type* key with a value corresponding to the *const char *type* provided by the VIF plug provider implementation as well as a *requested-chassis* key with a value pointing at the name or hostname of the chassis it wants the VIF plugged on. Additional VIF plug provider specific key/value pairs must be provided for successful lookup.
2. *ovn-northd* looks up the name or hostname provided in the *requested-chassis* option and fills the OVN Southbound Port_Binding requested_chassis column, it also copies relevant options over to the Port_Binding record.
3. *ovn-controller* monitors Southbound Port_Binding entries with a requested_chassis column pointing at its chassis UUID. When it encounters an entry with option *vif-plug-type* and it has registered a VIF plug provider matching that type, it will act on it even if no local binding exists yet.
4. It will fill the *struct vif_plug_port_ctx_in* as defined in *lib/vif-plug.h* with *op_type* set to 'PLUG_OP_CREATE' and make a call to the VIF plug providers *vif_plug_port_prepare* callback function. VIF plug provider performs lookup and fills the *struct vif_plug_port_ctx_out* as defined in *lib/vif-plug.h*.
5. *ovn-controller* creates a port and interface record in the local OVSDB using the details provided by the VIF plug provider and also adds *external-ids:iface-id* with value matching the logical port name and *external-ids:ovn-plugged* with value matching the logical port *vif-plug-type*. When the port creation is done a call will first be made to the VIF plug providers *vif_plug_port_finish* function and then to the *vif_plug_port_ctx_destroy* function to free any memory allocated by the VIF plug implementation.
6. The Open vSwitch vswitchd will assign a ofport to the newly created interface and on the next *ovn-controller* main loop iteration flows will be installed.
7. On each main loop iteration the *ovn-controller* will in addition to normal flow processing make a call to the VIF plug provider again similar to the first creation in case anything needs updating for the interface record.
8. The port will be unplugged when an event occurs which would make the *ovn-controller* release a logical port, for example the Logical_Switch_Port and Port_Binding entry disappearing from the database or its *requested_chassis* column being pointed to a different chassis.

The VIF plug provider interface

The interface between internals of OVN and a VIF plug provider is a set of callbacks as defined by the *struct vif_plug_class* in *lib/vif-plug-provider.h*.

It is important to note that these callbacks will be called in the critical path of the *ovn-controller* processing loop, so care must be taken to make the implementation as efficient as possible, and under no circumstance can any of the callback functions make calls that block.

On *ovn-controller* startup, VIF plug providers made available at build time will be registered by the identifier provided in the *const char *type* pointer, at this time the *init* function pointer will be called if it is non-NULL.

> **Note: apart from the *const char *type* pointer, no attempt will be made** to access VIF plug provider data or functions before the call to the *init* has been made.

On *ovn-controller* exit, the VIF plug providers registered in the above mentioned procedure will have their *destroy* function pointer called if it is non-NULL.

If the VIF plug provider has internal lookup tables that need to be maintained they can define a *run* function which will be called as part of the *ovn-controller* main loop. If there are any changes encountered the function should return 'true' to signal that further processing is necessary, 'false' otherwise.

On update of Interface records the *ovn-controller* will pass on a *sset* to the *ovsport_update_iface* function containing options the plug implementation finds pertinent to maintain for successful operation. This *sset* is retrieved by making a call to the plug implementation *vif_plug_get_maintained_iface_options* function pointer if it is non-NULL. This allows presence of other users of the OVSDB maintaining a different set of options on the same set of Interface records without wiping out their changes.

Before creating or updating an existing interface record the VIF plug provider *vif_plug_port_prepare* function pointer will be called with valid pointers to *struct vif_plug_port_ctx_in* and *struct vif_plug_port_ctx_out* data structures. If the VIF plug provider implementation is able to perform lookup it should fill the *struct vif_plug_port_ctx_out* data structure and return 'true'. The *ovn-controller* will then create or update the port/interface records and then call *vif_plug_port_finish* when the transactions commits and *vif_plug_port_ctx_destroy* to free any allocated memory. If the VIF plug provider implementation is unable to perform lookup or prepare the desired resource at this time, it should return 'false' which will tell the *ovn-controller* to not plug the port, in this case it will not call *vif_plug_port_finish* nor *vif_plug_port_ctx_destroy*.

> **Note: The VIF plug provider implementation should exhaust all** non-blocking options to succeed with lookup from within the *vif_plug_port_prepare* handler, including refreshing lookup tables if necessary.

Before removing port and interface records previously plugged by the *ovn-controller* as identified by presence of the Interface *external-ids:ovn-plugged* key, the *ovn-controller* will look up the *vif-plug-type* from *external-ids:ovn-plugged*, fill *struct vif_plug_port_ctx_in* with *op_type* set to 'PLUG_OP_REMOVE' and make a call to *vif_plug_port_prepare*. After the port and interface has been removed a call will be made to *vif_plug_port_finish*. Both calls will be made with the pointer to *vif_plug_port_ctx_out* set to 'NULL', and no call will be made to *vif_plug_port_ctx_destroy*.

Building with in-tree VIF plug providers

VIF plug providers hosted in the OVN repository live under *lib/vif-plug-providers*:

To enable them, provide the *–enable-vif-plug-providers* command line option to the configure script when building OVN.

Building with an externally provided VIF plug provider

There is also infrastructure in place to support linking OVN with an externally built VIF plug provider.

This external VIF plug provider must define a NULL-terminated array of pointers to *struct vif_plug_class* data structures named *vif_plug_provider_classes*. Example:

```
const struct vif_plug_class *vif_plug_provider_classes[] = {
    &vif_plug_foo,
    NULL,
};
```

The name of the repository for the external VIF plug provider should be the same as the name of the library it produces, and the built library artifact should be placed in lib/.libs. Example:

```
ovn-vif-foo/
ovn-vif-foo/lib/.libs/libovn-vif-foo.la
```

To enable such a VIF plug provider provide the *–with-vif-plug-provider=/path/to/ovn-vif-foo* command line option to the configure script when building OVN.

4.1.7 Testing

It is possible to test OVN using both tooling provided with Open vSwitch and using a variety of third party tooling.

Built-in Tooling

OVN provides a number of different test suites and other tooling for validating basic functionality of OVN. Before running any of the tests described here, you must bootstrap, configure and build OVN as described in *OVN on Linux, FreeBSD and NetBSD*. You do not need to install OVN, Open vSwitch or to build or load the kernel module to run these test suites.You do not need supervisor privilege to run these test suites.

Unit Tests

OVN includes a suite of self-tests. Before you submit patches upstream, we advise that you run the tests and ensure that they pass. If you add new features to OVN, then adding tests for those features will ensure your features don't break as developers modify other areas of OVN.

To run all the unit tests in OVN, one at a time, run:

```
$ make check
```

This takes under 5 minutes on a modern desktop system.

To run all the unit tests in OVN in parallel, run:

```
$ make check TESTSUITEFLAGS=-j8
```

You can run up to eight threads. This takes under a minute on a modern 4-core desktop system.

To see a list of all the available tests, run:

```
$ make check TESTSUITEFLAGS=--list
```

To run only a subset of tests, e.g. test 123 and tests 477 through 484, run:

```
$ make check TESTSUITEFLAGS='123 477-484'
```

Tests do not have inter-dependencies, so you may run any subset.

To run tests matching a keyword, e.g. `ovsdb`, run:

```
$ make check TESTSUITEFLAGS='-k ovsdb'
```

To see a complete list of test options, run:

```
$ make check TESTSUITEFLAGS=--help
```

The results of a testing run are reported in `tests/testsuite.log`. Report report test failures as bugs and include the `testsuite.log` in your report.

Note: Sometimes a few tests may fail on some runs but not others. This is usually a bug in the testsuite, not a bug in Open vSwitch itself. If you find that a test fails intermittently, please report it, since the developers may not have noticed. You can make the testsuite automatically rerun tests that fail, by adding `RECHECK=yes` to the `make` command line, e.g.:

```
$ make check TESTSUITEFLAGS=-j8 RECHECK=yes
```

Debugging unit tests

To initiate debugging from artifacts generated from *make check* run, set the `OVS_PAUSE_TEST` environment variable to 1. For example, to run test case 139 and pause on error:

```
$ OVS_PAUSE_TEST=1 make check TESTSUITEFLAGS='-v 139'
```

When error occurs, above command would display something like this:

```
Set environment variable to use various ovs utilities
export OVS_RUNDIR=<dir>/ovs/_build-gcc/tests/testsuite.dir/0139
Press ENTER to continue:
```

And from another window, one can execute ovs-xxx commands like:

```
export OVS_RUNDIR=/opt/vdasari/Developer/ovs/_build-gcc/tests/testsuite.dir/0139
$ ovs-ofctl dump-ports br0
.
.
```

Once done with investigation, press ENTER to perform cleanup operation.

Coverage

If the build was configured with `--enable-coverage` and the `lcov` utility is installed, you can run the testsuite and generate a code coverage report by using the `check-lcov` target:

```
$ make check-lcov
```

All the same options are available via TESTSUITEFLAGS. For example:

```
$ make check-lcov TESTSUITEFLAGS='-j8 -k ovn'
```

Valgrind

If you have `valgrind` installed, you can run the testsuite under valgrind by using the `check-valgrind` target:

```
$ make check-valgrind
```

When you do this, the "valgrind" results for test `<N>` are reported in files named `tests/testsuite.dir/<N>/valgrind.*`.

To test the testsuite of kernel datapath under valgrind, you can use the `check-kernel-valgrind` target and find the "valgrind" results under directory `tests/system-kmod-testsuite.dir/`.

All the same options are available via TESTSUITEFLAGS.

Hint: You may find that the valgrind results are easier to interpret if you put `-q` in `~/.valgrindrc`, since that reduces the amount of output.

Static Code Analysis

Static Analysis is a method of debugging Software by examining code rather than actually executing it. This can be done through 'scan-build' commandline utility which internally uses clang (or) gcc to compile the code and also invokes a static analyzer to do the code analysis. At the end of the build, the reports are aggregated in to a common folder and can later be analyzed using 'scan-view'.

OVN includes a Makefile target to trigger static code analysis:

```
$ ./boot.sh
$ ./configure CC=clang  # clang
# or
$ ./configure CC=gcc CFLAGS="-std=gnu99"  # gcc
$ make clang-analyze
```

You should invoke scan-view to view analysis results. The last line of output from `clang-analyze` will list the command (containing results directory) that you should invoke to view the results on a browser.

Continuous Integration with Travis CI

A .travis.yml file is provided to automatically build OVN with various build configurations and run the testsuite using Travis CI. Builds will be performed with gcc, sparse and clang with the -Werror compiler flag included, therefore the build will fail if a new warning has been introduced.

The CI build is triggered via git push (regardless of the specific branch) or pull request against any Open vSwitch GitHub repository that is linked to travis-ci.

Instructions to setup travis-ci for your GitHub repository:

1. Go to https://travis-ci.org/ and sign in using your GitHub ID.
2. Go to the "Repositories" tab and enable the ovs repository. You may disable builds for pushes or pull requests.
3. In order to avoid forks sending build failures to the upstream mailing list, the notification email recipient is encrypted. If you want to receive email notification for build failures, replace the the encrypted string:
    1. Install the travis-ci CLI (Requires ruby >=2.0): gem install travis
    2. In your Open vSwitch repository: travis encrypt mylist@mydomain.org

3. Add/replace the notifications section in .travis.yml and fill in the secure string as returned by travis encrypt:

```
notifications:
  email:
    recipients:
      - secure: "....."
```

Note: You may remove/omit the notifications section to fall back to default notification behaviour which is to send an email directly to the author and committer of the failing commit. Note that the email is only sent if the author/committer have commit rights for the particular GitHub repository.

4. Pushing a commit to the repository which breaks the build or the testsuite will now trigger a email sent to mylist@mydomain.org

Datapath testing

OVN includes a suite of tests specifically for datapath functionality. The datapath tests make some assumptions about the environment. They must be run under root privileges on a Linux system with support for network namespaces. Make sure no other Open vSwitch instance is running the test suite. These tests may take several minutes to complete, and cannot be run in parallel.

To invoke the datapath testsuite with the OVS userspace datapath, run:

```
$ make check-system-userspace
```

The results of the userspace testsuite appear in `tests/system-userspace-testsuite.dir`.

To invoke the datapath testsuite with the OVS kernel datapath, run:

```
$ make check-kernel
```

The results of the kernel testsuite appear in `tests/system-kmod-testsuite.dir`.

The tests themselves must run as root. If you do not run `make` as root, then you can specify a program to get superuser privileges as `SUDO=<program>`, e.g. the following uses `sudo` (the `-E` option is needed to pass through environment variables):

```
$ make check-system-userspace SUDO='sudo -E'
```

The testsuite creates and destroys tap devices named `ovs-netdev` and `br0`. If it is interrupted during a test, then before it can be restarted, you may need to destroy these devices with commands like the following:

```
$ ip tuntap del dev ovs-netdev mode tap
$ ip tuntap del dev br0 mode tap
```

All the features documented under *Unit Tests* are available for the datapath testsuites, except that the datapath testsuites do not support running tests in parallel.

Performance testing

OVN includes a suite of micro-benchmarks to aid a developer in understanding the performance impact of any changes that they are making. They can be used to help to understand the relative performance between two test runs on the same test machine, but are not intended to give the absolute performance of OVN.

To invoke the performance testsuite, run:

```
$ make check-perf
```

This will run all available performance tests. Some of these tests may be long-running as they need to build complex logical network topologies. In order to speed up subsequent test runs, some objects (e.g. the Northbound DB) may be cached. In order to force the tests to rebuild all these objects, run:

```
$ make check-perf TESTSUITEFLAGS="--rebuild"
```

A typical workflow for a developer trying to improve the performance of OVN would be the following:

0. Optional: Modify/add a performance test to buld the topology that you are benchmarking, if required.
1. Run `make check-perf TESTSUITEFLAGS="--rebuild"` to generate cached databases (and complete a test run). The results of each test run are displayed on the screen at the end of the test run but are also saved in the file `tests/perf-testsuite.dir/results`.

Note: This step may take some time depending on the number of tests that are being rebuilt, the complexity of the tests and the performance of the test machine. If you are only using one test, you can specify the test to run by adding the test number to the `make` command. (e.g. `make check-perf TESTSUITEFLAGS="--rebuild <test number>"`)

---

2. Run `make check-perf` to measure the performance metric that you are benchmarking against. If you are only using one test, you can specify the test to run by adding the test number to the `make` command. (e.g. `make check-perf TESTSUITEFLAGS="--rebuild <test number>"`)
3. Modify OVN code to implement the change that you believe will improve the performance.
4. Go to Step 2. to continue making improvements.

If, as a developer, you modify a performance test in a way that may change one of these cached objects, be sure to rebuild the test.

The cached objects are stored under the relevant folder in `tests/perf-testsuite.dir/cached`.

ovn-architecture(7)	(pdf)	(html)	(plain text)

CHAPTER 5

How-to Guides

Answers to common "How do I?"-style questions. For more information on the topics covered herein, refer to *Deep Dive*.

5.1 OVS

5.1.1 Encrypt Open vSwitch Tunnels with IPsec

Please refer to the Open vSwitch documentation on ipsec.

5.1.2 Open vSwitch with SSL

Please refer to the Open vSwitch documentation on SSL.

5.2 OVN

5.2.1 Open Virtual Networking With Docker

This document describes how to use Open Virtual Networking with Docker 1.9.0 or later.

Important: Requires Docker version 1.9.0 or later. Only Docker 1.9.0+ comes with support for multi-host networking. Consult www.docker.com for instructions on how to install Docker.

Note: You must build and install Open vSwitch before proceeding with the below guide. Refer to *Installing Open Virtual Network (OVN)* for more information.

Setup

For multi-host networking with OVN and Docker, Docker has to be started with a distributed key-value store. For example, if you decide to use consul as your distributed key-value store and your host IP address is `$HOST_IP`, start your Docker daemon with:

```
$ docker daemon --cluster-store=consul://127.0.0.1:8500 \
    --cluster-advertise=$HOST_IP:0
```

OVN provides network virtualization to containers. OVN's integration with Docker currently works in two modes - the "underlay" mode or the "overlay" mode.

In the "underlay" mode, OVN requires a OpenStack setup to provide container networking. In this mode, one can create logical networks and can have containers running inside VMs, standalone VMs (without having any containers running inside them) and physical machines connected to the same logical network. This is a multi-tenant, multi-host solution.

In the "overlay" mode, OVN can create a logical network amongst containers running on multiple hosts. This is a single-tenant (extendable to multi-tenants depending on the security characteristics of the workloads), multi-host solution. In this mode, you do not need a pre-created OpenStack setup.

For both the modes to work, a user has to install and start Open vSwitch in each VM/host that they plan to run their containers on.

The "overlay" mode

Note: OVN in "overlay" mode needs a minimum Open vSwitch version of 2.5.

1. Start the central components.

   OVN architecture has a central component which stores your networking intent in a database. On one of your machines, with an IP Address of `$CENTRAL_IP`, where you have installed and started Open vSwitch, you will need to start some central components.

   Start ovn-northd daemon. This daemon translates networking intent from Docker stored in the OVN_Northbound database to logical flows in `OVN_Southbound` database. For example:

   ```
   $ /usr/share/ovn/scripts/ovn-ctl start_northd
   ```

   With Open vSwitch version of 2.7 or greater, you need to run the following additional commands (Please read the manpages of ovn-nb for more control on the types of connection allowed.)

   ```
   $ ovn-nbctl set-connection ptcp:6641
   $ ovn-sbctl set-connection ptcp:6642
   ```

2. One time setup

   On each host, where you plan to spawn your containers, you will need to run the below command once. You may need to run it again if your OVS database gets cleared. It is harmless to run it again in any case:

   ```
   $ ovs-vsctl set Open_vSwitch . \
       external_ids:ovn-remote="tcp:$CENTRAL_IP:6642" \
       external_ids:ovn-nb="tcp:$CENTRAL_IP:6641" \
       external_ids:ovn-encap-ip=$LOCAL_IP \
       external_ids:ovn-encap-type="$ENCAP_TYPE"
   ```

where:

$LOCAL_IP is the IP address via which other hosts can reach this host. This acts as your local tunnel endpoint.

$ENCAP_TYPE is the type of tunnel that you would like to use for overlay networking. The options are `geneve` or `stt`. Your kernel must have support for your chosen `$ENCAP_TYPE`. Both `geneve` and `stt` are part of the Open vSwitch kernel module that is compiled from this repo. If you use the Open vSwitch kernel module from upstream Linux, you will need a minimum kernel version of 3.18 for `geneve`. There is no `stt` support in upstream Linux. You can verify whether you have the support in your kernel as follows:

```
$ lsmod | grep $ENCAP_TYPE
```

In addition, each Open vSwitch instance in an OVN deployment needs a unique, persistent identifier, called the `system-id`. If you install OVS from distribution packaging for Open vSwitch (e.g. .deb or .rpm packages), or if you use the ovs-ctl utility included with Open vSwitch, it automatically configures a system-id. If you start Open vSwitch manually, you should set one up yourself. For example:

```
$ id_file=/etc/openvswitch/system-id.conf
$ test -e $id_file || uuidgen > $id_file
$ ovs-vsctl set Open_vSwitch . external_ids:system-id=$(cat $id_file)
```

3. Start the `ovn-controller`.

   You need to run the below command on every boot:

```
$ /usr/share/ovn/scripts/ovn-ctl start_controller
```

4. Start the Open vSwitch network driver.

   By default Docker uses Linux bridge for networking. But it has support for external drivers. To use Open vSwitch instead of the Linux bridge, you will need to start the Open vSwitch driver.

   The Open vSwitch driver uses the Python's flask module to listen to Docker's networking api calls. So, if your host does not have Python's flask module, install it:

```
$ sudo pip install Flask
```

   Start the Open vSwitch driver on every host where you plan to create your containers. Refer to the note on `$OVS_PYTHON_LIBS_PATH` that is used below at the end of this document:

```
$ PYTHONPATH=$OVS_PYTHON_LIBS_PATH ovn-docker-overlay-driver --detach
```

Note: The `$OVS_PYTHON_LIBS_PATH` variable should point to the directory where Open vSwitch Python modules are installed. If you installed Open vSwitch Python modules via the Debian package of `python-openvswitch` or via pip by running `pip install ovs`, you do not need to specify the PATH. If you installed it by following the instructions in *OVN on Linux, FreeBSD and NetBSD*, then you should specify the PATH. In this case, the PATH depends on the options passed to `./configure`. It is usually either `/usr/share/openvswitch/python` or `/usr/local/share/openvswitch/python`

Docker has inbuilt primitives that closely match OVN's logical switches and logical port concepts. Consult Docker's documentation for all the possible commands. Here are some examples.

Create a logical switch

To create a logical switch with name 'foo', on subnet '192.168.1.0/24', run:

```
$ NID=`docker network create -d openvswitch --subnet=192.168.1.0/24 foo`
```

List all logical switches

```
$ docker network ls
```

You can also look at this logical switch in OVN's northbound database by running the following command:

```
$ ovn-nbctl --db=tcp:$CENTRAL_IP:6640 ls-list
```

Delete a logical switch

```
$ docker network rm bar
```

Create a logical port

Docker creates your logical port and attaches it to the logical network in a single step. For example, to attach a logical port to network `foo` inside container busybox, run:

```
$ docker run -itd --net=foo --name=busybox busybox
```

List all logical ports

Docker does not currently have a CLI command to list all logical ports but you can look at them in the OVN database by running:

```
$ ovn-nbctl --db=tcp:$CENTRAL_IP:6640 lsp-list $NID
```

Create and attach a logical port to a running container

```
$ docker network create -d openvswitch --subnet=192.168.2.0/24 bar
$ docker network connect bar busybox
```

Detach and delete a logical port from a running container

You can delete your logical port and detach it from a running container by running:

```
$ docker network disconnect bar busybox
```

The "underlay" mode

Note: This mode requires that you have a OpenStack setup pre-installed with OVN providing the underlay networking.

1. One time setup

   A OpenStack tenant creates a VM with a single network interface (or multiple) that belongs to management logical networks. The tenant needs to fetch the port-id associated with the interface via which he plans to send the container traffic inside the spawned VM. This can be obtained by running the below command to fetch the 'id' associated with the VM:

   ```
   $ nova list
   ```

   and then by running:

   ```
   $ neutron port-list --device_id=$id
   ```

   Inside the VM, download the OpenStack RC file that contains the tenant information (henceforth referred to as `openrc.sh`). Edit the file and add the previously obtained port-id information to the file by appending the following line:

   ```
   $ export OS_VIF_ID=$port_id
   ```

   After this edit, the file will look something like:

   ```
   #!/bin/bash
   export OS_AUTH_URL=http://10.33.75.122:5000/v2.0
   export OS_TENANT_ID=fab106b215d943c3bad519492278443d
   export OS_TENANT_NAME="demo"
   export OS_USERNAME="demo"
   export OS_VIF_ID=e798c371-85f4-4f2d-ad65-d09dd1d3c1c9
   ```

2. Create the Open vSwitch bridge

   If your VM has one ethernet interface (e.g.: 'eth0'), you will need to add that device as a port to an Open vSwitch bridge 'breth0' and move its IP address and route related information to that bridge. (If it has multiple network interfaces, you will need to create and attach an Open vSwitch bridge for the interface via which you plan to send your container traffic.)

   If you use DHCP to obtain an IP address, then you should kill the DHCP client that was listening on the physical Ethernet interface (e.g. eth0) and start one listening on the Open vSwitch bridge (e.g. breth0).

   Depending on your VM, you can make the above step persistent across reboots. For example, if your VM is Debian/Ubuntu-based, read *openvswitch-switch.README.Debian* found in *debian* folder. If your VM is RHEL-based, refer to *RHEL 5.6, 6.x Packaging for Open vSwitch*.

3. Start the Open vSwitch network driver

   The Open vSwitch driver uses the Python's flask module to listen to Docker's networking api calls. The driver also uses OpenStack's `python-neutronclient` libraries. If your host does not have Python's `flask` module or `python-neutronclient` you must install them. For example:

   ```
   $ pip install python-neutronclient
   $ pip install Flask
   ```

   Once installed, source the `openrc` file:

   ```
   $ . ./openrc.sh
   ```

   Start the network driver and provide your OpenStack tenant password when prompted:

   ```
   $ PYTHONPATH=$OVS_PYTHON_LIBS_PATH ovn-docker-underlay-driver \
       --bridge breth0 --detach
   ```

From here-on you can use the same Docker commands as described in *docker-overlay*.

Refer to the ovs-architecture man pages (`man ovn-architecture`) to understand OVN's architecture in detail.

5.2.2 Integration of Containers with OVN and OpenStack

Isolation between containers is weaker than isolation between VMs, so some environments deploy containers for different tenants in separate VMs as an additional security measure. This document describes creation of containers inside VMs and how they can be made part of the logical networks securely. The created logical network can include VMs, containers and physical machines as endpoints. To better understand the proposed integration of containers with OVN and OpenStack, this document describes the end to end workflow with an example.

- A OpenStack tenant creates a VM (say VM-A) with a single network interface that belongs to a management logical network. The VM is meant to host containers. OpenStack Nova chooses the hypervisor on which VM-A is created.
- A Neutron port may have been created in advance and passed in to Nova with the request to create a new VM. If not, Nova will issue a request to Neutron to create a new port. The ID of the logical port from Neutron will also be used as the vif-id for the virtual network interface (VIF) of VM-A.
- When VM-A is created on a hypervisor, its VIF gets added to the Open vSwitch integration bridge. This creates a row in the Interface table of the `Open_vSwitch` database. As explained in the *integration guide*, the vif-id associated with the VM network interface gets added in the `external_ids:iface-id` column of the newly created row in the Interface table.
- Since VM-A belongs to a logical network, it gets an IP address. This IP address is used to spawn containers (either manually or through container orchestration systems) inside that VM and to monitor the health of the created containers.
- The vif-id associated with the VM's network interface can be obtained by making a call to Neutron using tenant credentials.
- This flow assumes a component called a "container network plugin". If you take Docker as an example for containers, you could envision the plugin to be either a wrapper around Docker or a feature of Docker itself that understands how to perform part of this workflow to get a container connected to a logical network managed by Neutron. The rest of the flow refers to this logical component that does not yet exist as the "container network plugin".
- All the calls to Neutron will need tenant credentials. These calls can either be made from inside the tenant VM as part of a container network plugin or from outside the tenant VM (if the tenant is not comfortable using temporary Keystone tokens from inside the tenant VMs). For simplicity, this document explains the work flow using the former method.
- The container hosting VM will need Open vSwitch installed in it. The only work for Open vSwitch inside the VM is to tag network traffic coming from containers.
- When a container needs to be created inside the VM with a container network interface that is expected to be attached to a particular logical switch, the network plugin in that VM chooses any unused VLAN (This VLAN tag only needs to be unique inside that VM. This limits the number of container interfaces to 4096 inside a single VM). This VLAN tag is stripped out in the hypervisor by OVN and is only useful as a context (or metadata) for OVN.
- The container network plugin then makes a call to Neutron to create a logical port. In addition to all the inputs that a call to create a port in Neutron that are currently needed, it sends the vif-id and the VLAN tag as inputs.
- Neutron in turn will verify that the vif-id belongs to the tenant in question and then uses the OVN specific plugin to create a new row in the Logical_Switch_Port table of the OVN Northbound Database. Neutron responds back with an IP address and MAC address for that network interface. So Neutron becomes the IPAM system and provides unique IP and MAC addresses across VMs and containers in the same logical network.

- The Neutron API call above to create a logical port for the container could add a relatively significant amount of time to container creation. However, an optimization is possible here. Logical ports could be created in advance and reused by the container system doing container orchestration. Additional Neutron API calls would only be needed if the port needs to be attached to a different logical network.
- When a container is eventually deleted, the network plugin in that VM may make a call to Neutron to delete that port. Neutron in turn will delete the entry in the `Logical_Switch_Port` table of the OVN Northbound Database.

As an example, consider Docker containers. Since Docker currently does not have a network plugin feature, this example uses a hypothetical wrapper around Docker to make calls to Neutron.

- Create a Logical switch:

```
$ ovn-docker --cred=cca86bd13a564ac2a63ddf14bf45d37f create network LS1
```

  The above command will make a call to Neutron with the credentials to create a logical switch. The above is optional if the logical switch has already been created from outside the VM.

- List networks available to the tenant:

```
$ ovn-docker --cred=cca86bd13a564ac2a63ddf14bf45d37f list networks
```

- Create a container and attach a interface to the previously created switch as a logical port:

```
$ ovn-docker --cred=cca86bd13a564ac2a63ddf14bf45d37f --vif-id=$VIF_ID \
    --network=LS1 run -d --net=none ubuntu:14.04 /bin/sh -c \
    "while true; do echo hello world; sleep 1; done"
```

  The above command will make a call to Neutron with all the inputs it currently needs to create a logical port. In addition, it passes the $VIF_ID and a unused VLAN. Neutron will add that information in OVN and return back a MAC address and IP address for that interface. ovn-docker will then create a veth pair, insert one end inside the container as 'eth0' and the other end as a port of a local OVS bridge as an access port of the chosen VLAN.

5.2.3 Open Virtual Network With firewalld

firewalld is a service that allows for easy administration of firewalls. OVN ships with a set of service files that can be used with firewalld to allow for remote connections to the northbound and southbound databases.

This guide will describe how you can use these files with your existing firewalld setup. Setup and administration of firewalld is outside the scope of this document.

Installation

If you have installed OVN from an RPM, then the service files for firewalld will automatically be installed in `/usr/lib/firewalld/services`. Installation from RPM includes installation from the yum or dnf package managers.

If you have installed OVN from source, then from the top level source directory, issue the following commands to copy the firewalld service files:

```
$ cp rhel/usr_lib_firewalld_services_ovn-central-firewall-service.xml \
/etc/firewalld/services/
$ cp rhel/usr_lib_firewalld_services_ovn-host-firewall-service.xml \
/etc/firewalld/services/
```

Activation

Assuming you are already running firewalld, you can issue the following commands to enable the OVN services.

On the central server (the one running `ovn-northd`), issue the following:

```
$ firewall-cmd --zone=public --add-service=ovn-central-firewall-service
```

This will open TCP ports 6641 and 6642, allowing for remote connections to the northbound and southbound databases.

On the OVN hosts (the ones running `ovn-controller`), issue the following:

```
$ firewall-cmd --zone=public --add-service=ovn-host-firewall-service
```

This will open UDP port 6081, allowing for geneve traffic to flow between the controllers.

Variations

When installing the XML service files, you have the choice of copying them to `/etc/firewalld/services` or `/usr/lib/firewalld/services`. The former is recommend since the latter can be overwritten if firewalld is upgraded.

The above commands assumed your underlay network interfaces are in the "public" firewalld zone. If your underlay network interfaces are in a separate zone, then adjust the above commands accordingly.

The `--permanent` option may be passed to the above firewall-cmd invocations in order for the services to be permanently added to the firewalld configuration. This way it is not necessary to re-issue the commands each time the firewalld service restarts.

The ovn-host-firewall-service only opens port 6081. This is because the default protocol for OVN tunnels is geneve. If you are using a different encapsulation protocol, you will need to modify the XML service file to open the appropriate port(s). For VXLAN, open port 4789. For STT, open port 7471.

Recommendations

The firewalld service files included with the OVS repo are meant as a convenience for firewalld users. All that the service files do is to open the common ports used by OVN. No additional security is provided. To ensure a more secure environment, it is a good idea to do the following

- Use tools such as iptables or nftables to restrict access to known hosts.
- Use SSL for all remote connections to OVN databases.
- Use role-based access control for connections to the OVN southbound database.

CHAPTER 6

Reference Guide

6.1 Man Pages

The following man pages are written in rST and converted to roff at compile time:

6.1.1 ovn-sim

Synopsis

`ovn-sim` [*option*]... [*script*]...

Description

`ovn-sim` is a wrapper script that adds ovn related commands on top of `ovs-sim`.

`ovs-sim` provides a convenient environment for running one or more Open vSwitch instances and related software in a sandboxed simulation environment.

To use `ovn-sim`, first build Open vSwitch, then invoke it directly from the build directory, e.g.:

```
git clone https://github.com/openvswitch/ovs.git
cd ovs
./boot.sh && ./configure && make
cd ..
git clone https://github.com/ovn-org/ovn.git
cd ovn
./boot.sh && ./configure --with-ovs-source=${PWD}/../ovs
make
utilities/ovn-sim
```

See documentation on `ovs-sim` for info on simulator, including the parameters you can use.

OVN Commands

These commands interact with OVN, the Open Virtual Network.

ovn_start [***options***] Creates and initializes the central OVN databases (both `ovn-sb(5)` and `ovn-nb(5)`) and starts an instance of `ovsdb-server` for each one. Also starts an instance of `ovn-northd`.

The following options are available:

--nbdb-model ***model*** Uses the given database model for the northbound database. The *model* may be `standalone` (the default), `backup`, or `clustered`.

--nbdb-servers ***n*** For a clustered northbound database, the number of servers in the cluster. The default is 3.

--sbdb-model ***model*** Uses the given database model for the southbound database. The *model* may be `standalone` (the default), `backup`, or `clustered`.

--sbdb-servers ***n*** For a clustered southbound database, the number of servers in the cluster. The default is 3.

ovn_attach ***network bridge ip*** [***masklen***] First, this command attaches bridge to interconnection network network, just like `net_attach` *network bridge*. Second, it configures (simulated) IP address *ip* (with network mask length *masklen*, which defaults to 24) on *bridge*. Finally, it configures the Open vSwitch database to work with OVN and starts `ovn-controller`.

Examples

Simulating hypervisors, starting ovn controller (via ovn_attach) and adding a logical port on each one of them:

```
ovn_start
ovn-nbctl ls-add lsw0
net_add n1
for i in 0 1; do
    sim_add hv$i
    as hv$i
    ovs-vsctl add-br br-phys
    ovn_attach n1 br-phys 192.168.0.`expr $i + 1`
    ovs-vsctl add-port br-int vif$i -- \
        set Interface vif$i external-ids:iface-id=lp$i
    ovn-nbctl lsp-add lsw0 lp$i
    ovn-nbctl lsp-set-addresses lp$i f0:00:00:00:00:0$i
done
```

Here's a primitive OVN "scale test" (adjust the scale by changing `n` in the first line):

```
n=200; export n
ovn_start --sbdb-model=clustered
net_add n1
ovn-nbctl ls-add br0
for i in `seq $n`; do
    (sim_add hv$i
    as hv$i
    ovs-vsctl add-br br-phys
    y=$(expr $i / 256)
    x=$(expr $i % 256)
    ovn_attach n1 br-phys 192.168.$y.$x
    ovs-vsctl add-port br-int vif$i -- \
```

(continues on next page)

(continued from previous page)

```
        set Interface vif$i external-ids:iface-id=lp$i) &
    case $i in
        *50|*00) echo $i; wait ;;
    esac
done
wait
for i in `seq $n`; do
    yy=$(printf %02x $(expr $i / 256))
    xx=$(printf %02x $(expr $i % 256))
    ovn-nbctl lsp-add br0 lp$i
    ovn-nbctl lsp-set-addresses lp$i f0:00:00:00:$yy:$xx
done
```

When the scale test has finished initializing, you can watch the logical ports come up with a command like this:

```
watch 'for i in `seq $n`; do \
if test `ovn-nbctl lsp-get-up lp$i` != up; then echo $i; fi; done'
```

The remainder are still in roff format can be found below:

ovn-architecture(7)	(pdf)	(html)	(plain text)
ovn-controller(8)	(pdf)	(html)	(plain text)
ovn-controller-vtep(8)	(pdf)	(html)	(plain text)
ovn-ctl(8)	(pdf)	(html)	(plain text)
ovn-nb(5)	(pdf)	(html)	(plain text)
ovn-nbctl(8)	(pdf)	(html)	(plain text)
ovn-northd(8)	(pdf)	(html)	(plain text)
ovn-sb(5)	(pdf)	(html)	(plain text)
ovn-sbctl(8)	(pdf)	(html)	(plain text)
ovn-trace(8)	(pdf)	(html)	(plain text)

CHAPTER 7

Open Virtual Network (OVN) FAQ

7.1 Development

Q: How do I apply patches from email?

A: You can use `git am` on raw email contents, either from a file saved by or piped from an email client. In `mutt`, for example, when you are viewing a patch, you can apply it to the tree in ~/ovs by issuing the command `|cd ~/ovs && git am`. If you are an OVS committer, you might want to add `-s` to sign off on the patch as part of applying it. If you do this often, then you can make the keystrokes `,a` shorthand for it by adding the following line to your `.muttrc`:

macro index,pager ,a "<pipe-message>cd ~/ovs && git am -s" "apply patch"

`git am` has a problem with some email messages from the ovs-dev list for which the mailing list manager edits the From: address, replacing it by the list's own address. The mailing list manager must do this for messages whose sender's email domain has DMARC configured, because receivers will otherwise discard these messages when they do not come directly from the sender's email domain. This editing makes the patches look like they come from the mailing list instead of the author. To work around this problem, one can use the following wrapper script for `git am`:

```
#! /bin/sh
tmp=$(mktemp)
cat >$tmp
if grep '^From:.*via dev.*' "$tmp" >/dev/null 2>&1; then
   sed '/^From:.*via dev.*/d
        s/^[Rr]eply-[tT]o:/From:/' $tmp
else
   cat "$tmp"
fi | git am "$@"
rm "$tmp"
```

Another way to apply emailed patches is to use the `pwclient` program, which can obtain patches from patchwork and apply them directly. Download `pwclient` at https://patchwork.ozlabs.org/project/ovn/. You probably want to set up a `.pwclientrc` that looks something like this:

```
[options]
default=ovn
signoff=true

[ovn]
url=https://patchwork.ozlabs.org/xmlrpc/
```

After you install `pwclient`, you can apply a patch from patchwork with `pwclient git-am #`, where # is the patch's number. (This fails with certain patches that contain form-feeds, due to a limitation of the protocol underlying `pwclient`.)

Another way to apply patches directly from patchwork which supports applying patch series is to use the `git-pw` program. It can be obtained with `pip install git-pw`. Alternative installation instructions and general documentation can be found at https://patchwork.readthedocs.io/projects/git-pw/en/latest/. You need to use your ovn patchwork login or create one at https://patchwork.ozlabs.org/register/. The following can then be set on the command line with `git config` or through a `.gitconfig` like this:

```
[pw]
server=https://patchwork.ozlabs.org/api/1.0
project=ovn
username=<username>
password=<password>
```

Patch series can be listed with `git-pw series list` and applied with `git-pw series apply #`, where # is the series number. Individual patches can be applied with `git-pw patch apply #`, where # is the patch number.

7.2 General

Q: What is OVN?

A: OVN, the Open Virtual Network, is a system to support virtual network abstraction. OVN complements the existing capabilities of OVS to add native support for virtual network abstractions, such as virtual L2 and L3 overlays and security groups.

OVN is intended to be used by cloud management software (CMS). For details about the architecture of OVN, see the ovn-architecture manpage. Some high-level features offered by OVN include

- Distributed virtual routers
- Distributed logical switches
- Access Control Lists
- DHCP
- DNS server

Q: How can I try OVN?

A: The OVN source code can be built on a Linux system. You can build and experiment with OVN on any Linux machine. Packages for various Linux distributions are available on many platforms, including: Debian, Ubuntu, Fedora.

Q: Why does OVN use STT and Geneve instead of VLANs or VXLAN (or GRE)?

A: OVN implements a fairly sophisticated packet processing pipeline in "logical datapaths" that can implement switching or routing functionality. A logical datapath has an ingress pipeline and an egress

pipeline, and each of these pipelines can include logic based on packet fields as well as packet metadata such as the logical ingress and egress ports (the latter only in the egress pipeline).

The processing for a logical datapath can be split across hypervisors. In particular, when a logical ingress pipeline executes an "output" action, OVN passes the packet to the egress pipeline on the hypervisor (or, in the case of output to a logical multicast group, hypervisors) on which the logical egress port is located. If this hypervisor is not the same as the ingress hypervisor, then the packet has to be transmitted across a physical network.

This situation is where tunneling comes in. To send the packet to another hypervisor, OVN encapsulates it with a tunnel protocol and sends the encapsulated packet across the physical network. When the remote hypervisor receives the tunnel packet, it decapsulates it and passes it through the logical egress pipeline. To do so, it also needs the metadata, that is, the logical ingress and egress ports.

Thus, to implement OVN logical packet processing, at least the following metadata must pass across the physical network:

- Logical datapath ID, a 24-bit identifier. In Geneve, OVN uses the VNI to hold the logical datapath ID; in STT, OVN uses 24 bits of STT's 64-bit context ID.
- Logical ingress port, a 15-bit identifier. In Geneve, OVN uses an option to hold the logical ingress port; in STT, 15 bits of the context ID.
- Logical egress port, a 16-bit identifier. In Geneve, OVN uses an option to hold the logical egress port; in STT, 16 bits of the context ID.

See `ovn-architecture(7)`, under "Tunnel Encapsulations", for details.

Together, these metadata require 24 + 15 + 16 = 55 bits. GRE provides 32 bits, VXLAN provides 24, and VLAN only provides 12. Most notably, if logical egress pipelines do not match on the logical ingress port, thereby restricting the class of ACLs available to users, then this eliminates 15 bits, bringing the requirement down to 40 bits. At this point, one can choose to limit the size of the OVN logical network in various ways, e.g.:

- 16 bits of logical datapaths + 16 bits of logical egress ports. This combination fits within a 32-bit GRE tunnel key.
- 12 bits of logical datapaths + 12 bits of logical egress ports. This combination fits within a 24-bit VXLAN VNI.
- It's difficult to identify an acceptable compromise for a VLAN-based deployment.

These compromises wouldn't suit every site, since some deployments may need to allocate more bits to the datapath or egress port identifiers.

As a side note, OVN does support VXLAN for use with ASIC-based top of rack switches, using `ovn-controller-vtep(8)` and the OVSDB VTEP schema described in `vtep(5)`, but this limits the features available from OVN to the subset available from the VTEP schema.

Q: How can I contribute to the OVN Community?

A: You can start by joining the mailing lists and helping to answer questions. You can also suggest improvements to documentation. If you have a feature or bug you would like to work on, send a mail to one of the *mailing lists*.

CHAPTER 8

OVN Internals

Information for people who want to know more about the OVN project itself and how they might involved.

8.1 Contributing to OVN

The below guides provide information on contributing to OVN itself.

8.1.1 Submitting Patches

Send changes to OVN as patches to dev@openvswitch.org. One patch per email. More details are included below.

If you are using Git, then *git format-patch* takes care of most of the mechanics described below for you.

Before You Start

Before you send patches at all, make sure that each patch makes sense. In particular:

- A given patch should not break anything, even if later patches fix the problems that it causes. The source tree should still build and work after each patch is applied. (This enables *git bisect* to work best.)
- A patch should make one logical change. Don't make multiple, logically unconnected changes to disparate subsystems in a single patch.
- A patch that adds or removes user-visible features should also update the appropriate user documentation or manpages. Consider adding an item to NEWS for nontrivial changes. Check "Feature Deprecation Guidelines" section in this document if you intend to remove user-visible feature.

Testing is also important:

- Test a patch that modifies existing code with `make check` before submission. Refer to the "Unit Tests" in *Testing*, for more information. We also encourage running the kernel and userspace system tests.
- Consider testing a patch that adds or deletes files with `make distcheck` before submission.

- A patch that modifies Linux kernel code should be at least build-tested on various Linux kernel versions before submission. I suggest versions 3.10 and whatever the current latest release version is at the time.
- A patch that adds a new feature should add appropriate tests for the feature. A bug fix patch should preferably add a test that would fail if the bug recurs.

If you are using GitHub, then you may utilize the travis-ci.org CI build system by linking your GitHub repository to it. This will run some of the above tests automatically when you push changes to your repository. See the "Continuous Integration with Travis-CI" in *Testing* for details on how to set it up.

Email Subject

The subject line of your email should be in the following format:

> [PATCH ovn <n>/<m>] <area>: <summary>

Where:

[PATCH ovn]: indicates that this is the patch and it is targeted for OVN project. This is important since OVN and OVS are using same mailing lists for development. `ovn` word could be added manually or by using `git format-patch --subject-prefix="PATCH ovn" ...`. It might be useful to add following configuration to a local `.git/config`:

```
[format]
      subjectPrefix = "PATCH ovn"
```

<n>/<m>: indicates that this is the nth of a series of m patches. It helps reviewers to read patches in the correct order. You may omit this prefix if you are sending only one patch.

<area>: indicates the area of OVN to which the change applies (often the name of a source file or a directory). You may omit it if the change crosses multiple distinct pieces of code.

<summary>: briefly describes the change. Use the imperative form, e.g. "Force SNAT for multiple gateway routers." or "Fix daemon exit for bad datapaths or flows." Try to keep the summary short, about 50 characters wide.

The subject, minus the `[PATCH ovn <n>/<m>]` prefix, becomes the first line of the commit's change log message.

Description

The body of the email should start with a more thorough description of the change. This becomes the body of the commit message, following the subject. There is no need to duplicate the summary given in the subject.

Please limit lines in the description to 75 characters in width. That allows the description to format properly even when indented (e.g. by "git log" or in email quotations).

The description should include:

- The rationale for the change.
- Design description and rationale (but this might be better added as code comments).
- Testing that you performed (or testing that should be done but you could not for whatever reason).
- Tags (see below).

There is no need to describe what the patch actually changed, if the reader can see it for himself.

If the patch refers to a commit already in the OVN repository, please include both the commit number and the subject of the patch, e.g. 'commit 632d136c (vswitch: Remove restriction on datapath names.)'.

If you, the person sending the patch, did not write the patch yourself, then the very first line of the body should take the form `From: <author name> <author email>`, followed by a blank line. This will automatically cause the named author to be credited with authorship in the repository.

Tags

The description ends with a series of tags, written one to a line as the last paragraph of the email. Each tag indicates some property of the patch in an easily machine-parseable manner.

Please don't wrap a tag across multiple lines. If necessary, it's OK to have a tag extend beyond the customary maximum width of a commit message.

Examples of common tags follow.

`Signed-off-by: Author Name <author.name@email.address...>`

Informally, this indicates that Author Name is the author or submitter of a patch and has the authority to submit it under the terms of the license. The formal meaning is to agree to the Developer's Certificate of Origin (see below).

If the author and submitter are different, each must sign off. If the patch has more than one author, all must sign off.

Signed-off-by tags should be the last tags in the commit message. If the author (or authors) and submitter are different, the author tags should come first. More generally, occasionally a patch might pass through a chain of submitters, and in such a case the sign-offs should be arranged in chronological order.

```
Signed-off-by: Author Name <author.name@email.address...>
Signed-off-by: Submitter Name <submitter.name@email.address...>
```

`Co-authored-by: Author Name <author.name@email.address...>`

Git can only record a single person as the author of a given patch. In the rare event that a patch has multiple authors, one must be given the credit in Git and the others must be credited via Co-authored-by: tags. (All co-authors must also sign off.)

`Acked-by: Reviewer Name <reviewer.name@email.address...>`

Reviewers will often give an `Acked-by:` tag to code of which they approve. It is polite for the submitter to add the tag before posting the next version of the patch or applying the patch to the repository. Quality reviewing is hard work, so this gives a small amount of credit to the reviewer.

Not all reviewers give `Acked-by:` tags when they provide positive reviews. It's customary only to add tags from reviewers who actually provide them explicitly.

`Tested-by: Tester Name <reviewer.name@email.address...>`

When someone tests a patch, it is customary to add a Tested-by: tag indicating that. It's rare for a tester to actually provide the tag; usually the patch submitter makes the tag himself in response to an email indicating successful testing results.

`Tested-at: <URL>`

When a test report is publicly available, this provides a way to reference it. Typical <URL>s would be build logs from autobuilders or references to mailing list archives.

Some autobuilders only retain their logs for a limited amount of time. It is less useful to cite these because they may be dead links for a developer reading the commit message months or years later.

`Reported-by: Reporter Name <reporter.name@email.address...>`

When a patch fixes a bug reported by some person, please credit the reporter in the commit log in this fashion. Please also add the reporter's name and email address to the list of people who provided helpful bug reports in the AUTHORS file at the top of the source tree.

Fairly often, the reporter of a bug also tests the fix. Occasionally one sees a combined "Reported-and-tested-by:" tag used to indicate this. It is also acceptable, and more common, to include both tags separately.

(If a bug report is received privately, it might not always be appropriate to publicly credit the reporter. If in doubt, please ask the reporter.)

`Requested-by: Requester Name <requester.name@email.address...>`

When a patch implements a request or a suggestion made by some person, please credit that person in the commit log in this fashion. For a helpful suggestion, please also add the person's name and email address to the list of people who provided suggestions in the AUTHORS file at the top of the source tree.

(If a suggestion or a request is received privately, it might not always be appropriate to publicly give credit. If in doubt, please ask.)

`Suggested-by: Suggester Name <suggester.name@email.address...>`

See `Requested-by:`.

`CC: Person <name@email>`

This is a way to tag a patch for the attention of a person when no more specific tag is appropriate. One use is to request a review from a particular person. It doesn't make sense to include the same person in CC and another tag, so e.g. if someone who is CCed later provides an Acked-by, add the Acked-by and remove the CC at the same time.

`Reported-at: <URL>`

If a patch fixes or is otherwise related to a bug reported in a public bug tracker, please include a reference to the bug in the form of a URL to the specific bug, e.g.:

```
Reported-at: https://bugs.debian.org/743635
```

This is also an appropriate way to refer to bug report emails in public email archives, e.g.:

```
Reported-at: https://mail.openvswitch.org/pipermail/ovs-dev/2014-June/284495.
↪html
```

`Submitted-at: <URL>`

If a patch was submitted somewhere other than the OVN development mailing list, such as a GitHub pull request, this header can be used to reference the source.

```
Submitted-at: https://github.com/openvswitch/ovs/pull/92
```

`VMware-BZ: #1234567`

If a patch fixes or is otherwise related to a bug reported in a private bug tracker, you may include some tracking ID for the bug for your own reference. Please include some identifier to make the origin clear, e.g. "VMware-BZ" refers to VMware's internal Bugzilla instance and "ONF-JIRA" refers to the Open Networking Foundation's JIRA bug tracker.

`ONF-JIRA: EXT-12345`

See `VMware-BZ:`.

`Bug #1234567.`

These are obsolete forms of VMware-BZ: that can still be seen in old change log entries. (They are obsolete because they do not tell the reader what bug tracker is referred to.)

`Issue: 1234567`

See `Bug:`.

`Fixes: 63bc9fb1c69f ("packets: Reorder CS_* flags to remove gap.")`

If you would like to record which commit introduced a bug being fixed, you may do that with a "Fixes" header. This assists in determining which OVN releases have the bug, so the patch can be applied to all affected versions. The easiest way to generate the header in the proper format is with this git command. This command also CCs the author of the commit being fixed, which makes sense unless the author also made the fix or is already named in another tag:

```
$ git log -1 --pretty=format:"CC: %an <%ae>%nFixes: %h (\"%s\")" \
  --abbrev=12 COMMIT_REF
```

`Vulnerability: CVE-2016-2074`

Specifies that the patch fixes or is otherwise related to a security vulnerability with the given CVE identifier. Other identifiers in public vulnerability databases are also suitable.

If the vulnerability was reported publicly, then it is also appropriate to cite the URL to the report in a Reported-at tag. Use a Reported-by tag to acknowledge the reporters.

Developer's Certificate of Origin

To help track the author of a patch as well as the submission chain, and be clear that the developer has authority to submit a patch for inclusion in OVN please sign off your work. The sign off certifies the following:

```
Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

(a) The contribution was created in whole or in part by me and I
    have the right to submit it under the open source license
    indicated in the file; or

(b) The contribution is based upon previous work that, to the best
    of my knowledge, is covered under an appropriate open source
    license and I have the right under that license to submit that
    work with modifications, whether created in whole or in part
    by me, under the same open source license (unless I am
    permitted to submit under a different license), as indicated
    in the file; or

(c) The contribution was provided directly to me by some other
    person who certified (a), (b) or (c) and I have not modified
    it.

(d) I understand and agree that this project and the contribution
    are public and that a record of the contribution (including all
    personal information I submit with it, including my sign-off) is
    maintained indefinitely and may be redistributed consistent with
    this project or the open source license(s) involved.
```

See also http://developercertificate.org/.

Feature Deprecation Guidelines

OVN is intended to be user friendly. This means that under normal circumstances we don't abruptly remove features from OVN that some users might still be using. Otherwise, if we would, then we would possibly break our user setup when they upgrade and would receive bug reports.

Typical process to deprecate a feature in OVN is to:

(a) Mention deprecation of a feature in the NEWS file. Also, mention expected release or absolute time when this feature would be removed from OVN altogether. Don't use relative time (e.g. "in 6 months") because that is not clearly interpretable.

(b) If OVN is configured to use deprecated feature it should print a warning message to the log files clearly indicating that feature is deprecated and that use of it should be avoided.

(c) If this feature is mentioned in man pages, then add "Deprecated" keyword to it.

Also, if there is alternative feature to the one that is about to be marked as deprecated, then mention it in (a), (b) and (c) as well.

Remember to follow-up and actually remove the feature from OVN codebase once deprecation grace period has expired and users had opportunity to use at least one OVN release that would have informed them about feature deprecation!

OVN upgrades

If the patch introduces any new OVN actions or updates existing OVN actions, then make sure to check the function ovn_get_internal_version() in lib/ovn-util.c and increment the macro - OVN_INTERNAL_MINOR_VER.

Adding new OVN actions or changing existing OVN actions can have datapath disruptions during OVN upgrades. To minimize disruptions, OVN supports version matching between ovn-northd and ovn-controller and it is important to update the internal OVN version when the patch introduces such changes.

Comments

If you want to include any comments in your email that should not be part of the commit's change log message, put them after the description, separated by a line that contains just `---`. It may be helpful to include a diffstat here for changes that touch multiple files.

Patch

The patch should be in the body of the email following the description, separated by a blank line.

Patches should be in `diff -up` format. We recommend that you use Git to produce your patches, in which case you should use the `-M -C` options to `git diff` (or other Git tools) if your patch renames or copies files. Quilt might be useful if you do not want to use Git.

Patches should be inline in the email message. Some email clients corrupt white space or wrap lines in patches. There are hints on how to configure many email clients to avoid this problem on kernel.org. If you cannot convince your email client not to mangle patches, then sending the patch as an attachment is a second choice.

Follow the style used in the code that you are modifying. *OVN Coding Style* file describes the coding style used in most of OVN.

You may use the `utilities/checkpatch.py` utility as a quick check for certain commonly occurring mistakes (improper leading/trailing whitespace, missing signoffs, some improper formatted patch files).

Example

```
From fa29a1c2c17682879e79a21bb0cdd5bbe67fa7c0 Mon Sep 17 00:00:00 2001
From: Jesse Gross <jesse@nicira.com>
Date: Thu, 8 Dec 2011 13:17:24 -0800
Subject: [PATCH] datapath: Alphabetize include/net/ipv6.h compat header.

Signed-off-by: Jesse Gross <jesse@nicira.com>
---
 datapath/linux/Modules.mk |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/datapath/linux/Modules.mk b/datapath/linux/Modules.mk
index fdd952e..f6cb88e 100644
--- a/datapath/linux/Modules.mk
+++ b/datapath/linux/Modules.mk
@@ -56,11 +56,11 @@ openvswitch_headers += \
    linux/compat/include/net/dst.h \
    linux/compat/include/net/genetlink.h \
    linux/compat/include/net/ip.h \
+   linux/compat/include/net/ipv6.h \
    linux/compat/include/net/net_namespace.h \
    linux/compat/include/net/netlink.h \
    linux/compat/include/net/protocol.h \
    linux/compat/include/net/route.h \
-   linux/compat/include/net/ipv6.h \
    linux/compat/genetlink.inc

 both_modules += brcompat
--
1.7.7.3
```

8.1.2 Backporting patches

Note: This is an advanced topic for developers and maintainers. Readers should familiarize themselves with building and running OVN, with the git tool, and with the OVN patch submission process.

The backporting of patches from one git tree to another takes multiple forms within OVN, but is broadly applied in the following fashion:

- Contributors submit their proposed changes to the latest development branch
- Contributors and maintainers provide feedback on the patches
- When the change is satisfactory, maintainers apply the patch to the development branch.
- Maintainers backport changes from a development branch to release branches.

With regards to OVN user space code and code that does not comprise the Linux datapath and compat code, the development branch is *master* in the OVN repository. Patches are applied first to this branch, then to the most recent *branch-X.Y*, then earlier *branch-X.Z*, and so on. The most common kind of patch in this category is a bugfix which affects master and other branches.

Changes to userspace components

Patches which are fixing bugs should be considered for backporting from *master* to release branches. OVN contributors submit their patches targeted to the *master* branch, using the `Fixes` tag described in *Submitting Patches*. The maintainer first applies the patch to *master*, then backports the patch to each older affected tree, as far back as it goes or at least to all currently supported branches. This is usually each branch back to the most recent LTS release branch.

If the fix only affects a particular branch and not *master*, contributors should submit the change with the target branch listed in the subject line of the patch. Contributors should list all versions that the bug affects. The `git format-patch` argument `--subject-prefix` may be used when posting the patch, for example:

```
$ git format-patch -1 --subject-prefix="PATCH ovn branch-21.06"
```

If a maintainer is backporting a change to older branches and the backport is not a trivial cherry-pick, then the maintainer may opt to submit the backport for the older branch on the mailing list for further review. This should be done in the same manner as described above.

Submission

Once the patches are all assembled and working on the OVN tree, they need to be formatted again using `git format-patch`. The common format for commit messages for Linux backport patches is as follows:

```
datapath: Remove incorrect WARN_ONCE().

Upstream commit:
    commit c6b2aafffc6934be72d96855c9a1d88970597fbc
    Author: Jarno Rajahalme <jarno@ovn.org>
    Date:   Mon Aug 1 19:08:29 2016 -0700

    openvswitch: Remove incorrect WARN_ONCE().

    ovs_ct_find_existing() issues a warning if an existing conntrack entry
    classified as IP_CT_NEW is found, with the premise that this should
    not happen.  However, a newly confirmed, non-expected conntrack entry
    remains IP_CT_NEW as long as no reply direction traffic is seen.  This
    has resulted into somewhat confusing kernel log messages.  This patch
    removes this check and warning.

    Fixes: 289f2253 ("openvswitch: Find existing conntrack entry after upcall.")
    Suggested-by: Joe Stringer <joe@ovn.org>
    Signed-off-by: Jarno Rajahalme <jarno@ovn.org>
    Acked-by: Joe Stringer <joe@ovn.org>

Signed-off-by: Jarno Rajahalme <jarno@ovn.org>
```

The upstream commit SHA should be the one that appears in Linus' tree so that reviewers can compare the backported patch with the one upstream. Note that the subject line for the backported patch replaces the original patch's `openvswitch` prefix with `datapath`. Patches which only affect the `datapath/linux/compat` directory should be prefixed with `compat`.

The contents of a backport should be equivalent to the changes made by the original patch; explain any variations from the original patch in the commit message - For instance if you rolled in a bugfix. Reviewers will verify that the changes made by the backport patch are the same as the changes made in the original commit which the backport is based upon. Patch submission should otherwise follow the regular steps described in *Submitting Patches*.

8.1.3 OVN Coding Style

This file describes the coding style used in C files in the OVN distribution.

The following GNU indent options approximate this style.

```
-npro -bad -bap -bbb -br -blf -brs -cdw -ce -fca -cli0 -npcs -i4 -l79 \
-lc79 -nbfda -nut -saf -sai -saw -sbi4 -sc -sob -st -ncdb -pi4 -cs -bs \
-di1 -lp -il0 -hnl
```

Basics

- Limit lines to 79 characters.
- Use form feeds (control+L) to divide long source files into logical pieces. A form feed should appear as the only character on a line.
- Do not use tabs for indentation.
- Avoid trailing spaces on lines.

Naming

- Use names that explain the purpose of a function or object.
- Use underscores to separate words in an identifier: `multi_word_name`.
- Use lowercase for most names. Use uppercase for macros, macro parameters, and members of enumerations.
- Give arrays names that are plural.
- Pick a unique name prefix (ending with an underscore) for each module, and apply that prefix to all of that module's externally visible names. Names of macro parameters, struct and union members, and parameters in function prototypes are not considered externally visible for this purpose.
- Do not use names that begin with _. If you need a name for "internal use only", use ___ as a suffix instead of a prefix.
- Avoid negative names: `found` is a better name than `not_found`.
- In names, a `size` is a count of bytes, a `length` is a count of characters. A buffer has size, but a string has length. The length of a string does not include the null terminator, but the size of the buffer that contains the string does.

Comments

Comments should be written as full sentences that start with a capital letter and end with a period. Put two spaces between sentences.

Write block comments as shown below. You may put the `/*` and `*/` on the same line as comment text if you prefer.

```
/*
 * We redirect stderr to /dev/null because we often want to remove all
 * traffic control configuration on a port so its in a known state.  If
 * this done when there is no such configuration, tc complains, so we just
 * always ignore it.
 */
```

Each function and each variable declared outside a function, and each struct, union, and typedef declaration should be preceded by a comment. See *functions* below for function comment guidelines.

Each struct and union member should each have an inline comment that explains its meaning. structs and unions with many members should be additionally divided into logical groups of members by block comments, e.g.:

```
/* An event that will wake the following call to poll_block(). */
struct poll_waiter {
    /* Set when the waiter is created. */
    struct ovs_list node;       /* Element in global waiters list. */
    int fd;                     /* File descriptor. */
    short int events;           /* Events to wait for (POLLIN, POLLOUT). */
    poll_fd_func *function;     /* Callback function, if any, or null. */
    void *aux;                  /* Argument to callback function. */
    struct backtrace *backtrace; /* Event that created waiter, or null. */

    /* Set only when poll_block() is called. */
    struct pollfd *pollfd;      /* Pointer to element of the pollfds array
                                   (null if added from a callback). */
};
```

Use `XXX` or `FIXME` comments to mark code that needs work.

Don't use `//` comments.

Don't comment out or `#if 0` out code. Just remove it. The code that was there will still be in version control history.

Functions

Put the return type, function name, and the braces that surround the function's code on separate lines, all starting in column 0.

Before each function definition, write a comment that describes the function's purpose, including each parameter, the return value, and side effects. References to argument names should be given in single-quotes, e.g. `'arg'`. The comment should not include the function name, nor need it follow any formal structure. The comment does not need to describe how a function does its work, unless this information is needed to use the function correctly (this is often better done with comments *inside* the function).

Simple static functions do not need a comment.

Within a file, non-static functions should come first, in the order that they are declared in the header file, followed by static functions. Static functions should be in one or more separate pages (separated by form feed characters) in logical groups. A commonly useful way to divide groups is by "level", with high-level functions first, followed by groups of progressively lower-level functions. This makes it easy for the program's reader to see the top-down structure by reading from top to bottom.

All function declarations and definitions should include a prototype. Empty parentheses, e.g. `int foo();`, do not include a prototype (they state that the function's parameters are unknown); write `void` in parentheses instead, e.g. `int foo(void);`.

Prototypes for static functions should either all go at the top of the file, separated into groups by blank lines, or they should appear at the top of each page of functions. Don't comment individual prototypes, but a comment on each group of prototypes is often appropriate.

In the absence of good reasons for another order, the following parameter order is preferred. One notable exception is that data parameters and their corresponding size parameters should be paired.

1. The primary object being manipulated, if any (equivalent to the `this` pointer in C++).
2. Input-only parameters.

3. Input/output parameters.
4. Output-only parameters.
5. Status parameter.

Example:

```
```
/* Stores the features supported by 'netdev' into each of '*current',
 * '*advertised', '*supported', and '*peer' that are non-null. Each value
 * is a bitmap of "enum ofp_port_features" bits, in host byte order.
 * Returns 0 if successful, otherwise a positive errno value. On failure,
 * all of the passed-in values are set to 0. */
int
netdev_get_features(struct netdev *netdev,
                    uint32_t *current, uint32_t *advertised,
                    uint32_t *supported, uint32_t *peer)
{
    ...
}
```
```

Functions that destroy an instance of a dynamically-allocated type should accept and ignore a null pointer argument. Code that calls such a function (including the C standard library function `free()`) should omit a null-pointer check. We find that this usually makes code easier to read.

Functions in `.c` files should not normally be marked `inline`, because it does not usually help code generation and it does suppress compiler warnings about unused functions. (Functions defined in `.h` usually should be marked `inline`.)

Function Prototypes

Put the return type and function name on the same line in a function prototype:

```
static const struct option_class *get_option_class(int code);
```

Omit parameter names from function prototypes when the names do not give useful information, e.g.:

```
int netdev_get_mtu(const struct netdev *, int *mtup);
```

Statements

Indent each level of code with 4 spaces. Use BSD-style brace placement:

```
if (a()) {
    b();
    d();
}
```

Put a space between `if`, `while`, `for`, etc. and the expressions that follow them.

Enclose single statements in braces:

```
if (a > b) {
    return a;
} else {
```

(continues on next page)

(continued from previous page)

```
    return b;
}
```

Use comments and blank lines to divide long functions into logical groups of statements.

Avoid assignments inside `if` and `while` conditions.

Do not put gratuitous parentheses around the expression in a return statement, that is, write `return 0;` and not `return(0);`

Write only one statement per line.

Indent `switch` statements like this:

```
switch (conn->state) {
case S_RECV:
    error = run_connection_input(conn);
    break;

case S_PROCESS:
    error = 0;
    break;

case S_SEND:
    error = run_connection_output(conn);
    break;

default:
    OVS_NOT_REACHED();
}
```

`switch` statements with very short, uniform cases may use an abbreviated style:

```
switch (code) {
case 200: return "OK";
case 201: return "Created";
case 202: return "Accepted";
case 204: return "No Content";
default: return "Unknown";
}
```

Use `for (;;)` to write an infinite loop.

In an `if/else` construct where one branch is the "normal" or "common" case and the other branch is the "uncommon" or "error" case, put the common case after the `if`, not the `else`. This is a form of documentation. It also places the most important code in sequential order without forcing the reader to visually skip past less important details. (Some compilers also assume that the `if` branch is the more common case, so this can be a real form of optimization as well.)

Return Values

For functions that return a success or failure indication, prefer one of the following return value conventions:

- An `int` where `0` indicates success and a positive errno value indicates a reason for failure.
- A `bool` where `true` indicates success and `false` indicates failure.

Macros

Don't define an object-like macro if an enum can be used instead.

Don't define a function-like macro if a `static inline` function can be used instead.

If a macro's definition contains multiple statements, enclose them with `do { ... } while (0)` to allow them to work properly in all syntactic circumstances.

Do use macros to eliminate the need to update different parts of a single file in parallel, e.g. a list of enums and an array that gives the name of each enum. For example:

```
/* Logging importance levels. */
#define VLOG_LEVELS                             \
    VLOG_LEVEL(EMER, LOG_ALERT)                 \
    VLOG_LEVEL(ERR, LOG_ERR)                    \
    VLOG_LEVEL(WARN, LOG_WARNING)               \
    VLOG_LEVEL(INFO, LOG_NOTICE)                \
    VLOG_LEVEL(DBG, LOG_DEBUG)
enum vlog_level {
#define VLOG_LEVEL(NAME, SYSLOG_LEVEL) VLL_##NAME,
    VLOG_LEVELS
#undef VLOG_LEVEL
    VLL_N_LEVELS
};

/* Name for each logging level. */
static const char *level_names[VLL_N_LEVELS] = {
#define VLOG_LEVEL(NAME, SYSLOG_LEVEL) #NAME,
    VLOG_LEVELS
#undef VLOG_LEVEL
};
```

Thread Safety Annotations

Use the macros in `lib/compiler.h` to annotate locking requirements. For example:

```
static struct ovs_mutex mutex = OVS_MUTEX_INITIALIZER;
static struct ovs_rwlock rwlock = OVS_RWLOCK_INITIALIZER;

void function_require_plain_mutex(void) OVS_REQUIRES(mutex);
void function_require_rwlock(void) OVS_REQ_RDLOCK(rwlock);
```

Pass lock objects, not their addresses, to the annotation macros. (Thus we have `OVS_REQUIRES(mutex)` above, not `OVS_REQUIRES(&mutex)`.)

Source Files

Each source file should state its license in a comment at the very top, followed by a comment explaining the purpose of the code that is in that file. The comment should explain how the code in the file relates to code in other files. The goal is to allow a programmer to quickly figure out where a given module fits into the larger system.

The first non-comment line in a `.c` source file should be:

```
#include <config.h>
```

`#include` directives should appear in the following order:

1. `#include <config.h>`
2. The module's own headers, if any. Including this before any other header (besides) ensures that the module's header file is self-contained (see *header files* below).
3. Standard C library headers and other system headers, preferably in alphabetical order. (Occasionally one encounters a set of system headers that must be included in a particular order, in which case that order must take precedence.)
4. OVN headers, in alphabetical order. Use `""`, not <>, to specify OVN header names.

Header Files

Each header file should start with its license, as described under *source files* above, followed by a "header guard" to make the header file idempotent, like so:

```
#ifndef NETDEV_H
#define NETDEV_H 1

...

#endif /* netdev.h */
```

Header files should be self-contained; that is, they should `#include` whatever additional headers are required, without requiring the client to `#include` them for it.

Don't define the members of a struct or union in a header file, unless client code is actually intended to access them directly or if the definition is otherwise actually needed (e.g. inline functions defined in the header need them).

Similarly, don't `#include` a header file just for the declaration of a struct or union tag (e.g. just for `struct ;`). Just declare the tag yourself. This reduces the number of header file dependencies.

Types

Use typedefs sparingly. Code is clearer if the actual type is visible at the point of declaration. Do not, in general, declare a typedef for a `struct`, `union`, or `enum`. Do not declare a typedef for a pointer type, because this can be very confusing to the reader.

A function type is a good use for a typedef because it can clarify code. The type should be a function type, not a pointer-to-function type. That way, the typedef name can be used to declare function prototypes. (It cannot be used for function definitions, because that is explicitly prohibited by C89 and C99.)

You may assume that `char` is exactly 8 bits and that `int` and `long` are at least 32 bits.

Don't assume that `long` is big enough to hold a pointer. If you need to cast a pointer to an integer, use `intptr_t` or `uintptr_t` from .

Use the `int_t` and `uint_t` types from for exact-width integer types. Use the `PRId`, `PRIu`, and `PRIx` macros from for formatting them with `printf()` and related functions.

For compatibility with antique `printf()` implementations:

- Instead of `"%zu"`, use `"%"PRIuSIZE`.
- Instead of `"%td"`, use `"%"PRIdPTR`.
- Instead of `"%ju"`, use `"%"PRIuMAX`.

Other variants exist for different radixes. For example, use `"%"PRIxSIZE` instead of `"%zx"` or `"%x"` instead of `"%hhx"`.

Also, instead of `"%hhd"`, use `"%d"`. Be cautious substituting `"%u"`, `"%x"`, and `"%o"` for the corresponding versions with `"hh"`: cast the argument to unsigned char if necessary, because `printf("%hhu", -1)` prints `255` but `printf("%u", -1)` prints `4294967295`.

Use bit-fields sparingly. Do not use bit-fields for layout of network protocol fields or in other circumstances where the exact format is important.

Declare bit-fields to be signed or unsigned integer types or `_Bool` (aka `bool`). Do *not* declare bit-fields of type `int`: C99 allows these to be either signed or unsigned according to the compiler's whim. (A 1-bit bit-field of type `int` may have a range of `-1...0`!)

Try to order structure members such that they pack well on a system with 2-byte `short`, 4-byte `int`, and 4- or 8-byte `long` and pointer types. Prefer clear organization over size optimization unless you are convinced there is a size or speed benefit.

Pointer declarators bind to the variable name, not the type name. Write `int *x`, not `int* x` and definitely not `int * x`.

Expressions

Put one space on each side of infix binary and ternary operators:

```
* / %
+ -
<< >>
< <= > >=
== !=
&
^
|
&&
||
?:
= += -= *= /= %= &= ^= |= <<= >>=
```

Avoid comma operators.

Do not put any white space around postfix, prefix, or grouping operators:

```
() [] -> .
! ~ ++ -- + - * &
```

Exception 1: Put a space after (but not before) the "sizeof" keyword.

Exception 2: Put a space between the `()` used in a cast and the expression whose type is cast: `(void *) 0`.

Break long lines before the ternary operators `?` and `:`, rather than after them, e.g.

```
return (out_port != VIGP_CONTROL_PATH
        ? alpheus_output_port(dp, skb, out_port)
        : alpheus_output_control(dp, skb, fwd_save_skb(skb),
                                 VIGR_ACTION));
```

Parenthesize the operands of `&&` and `||` if operator precedence makes it necessary, or if the operands are themselves expressions that use `&&` and `||`, but not otherwise. Thus:

```
if (rule && (!best || rule->priority > best->priority)) {
    best = rule;
}
```

but:

```
if (!isdigit((unsigned char)s[0]) ||
    !isdigit((unsigned char)s[1]) ||
    !isdigit((unsigned char)s[2])) {
    printf("string %s does not start with 3-digit code\n", s);
}
```

Do parenthesize a subexpression that must be split across more than one line, e.g.:

```
*idxp = ((l1_idx << PORT_ARRAY_L1_SHIFT) |
         (l2_idx << PORT_ARRAY_L2_SHIFT) |
         (l3_idx << PORT_ARRAY_L3_SHIFT));
```

Breaking a long line after a binary operator gives its operands a more consistent look, since each operand has the same horizontal position. This makes the end-of-line position a good choice when the operands naturally resemble each other, as in the previous two examples. On the other hand, breaking before a binary operator better draws the eye to the operator, which can help clarify code by making it more obvious what's happening, such as in the following example:

```
if (!ctx.freezing
    && xbridge->has_in_band
    && in_band_must_output_to_local_port(flow)
    && !actions_output_to_local_port(&ctx)) {
```

Thus, decide whether to break before or after a binary operator separately in each situation, based on which of these factors appear to be more important.

Try to avoid casts. Don't cast the return value of malloc().

The `sizeof` operator is unique among C operators in that it accepts two very different kinds of operands: an expression or a type. In general, prefer to specify an expression, e.g. `int *x = xmalloc(sizeof *x);`. When the operand of `sizeof` is an expression, there is no need to parenthesize that operand, and please don't.

Use the `ARRAY_SIZE` macro from `lib/util.h` to calculate the number of elements in an array.

When using a relational operator like < or ==, put an expression or variable argument on the left and a constant argument on the right, e.g. `x == 0`, *not* `0 == x`.

Blank Lines

Put one blank line between top-level definitions of functions and global variables.

C DIALECT

Most C99 features are OK because they are widely implemented:

- Flexible array members (e.g. `struct { int foo[]; }`).
- `static inline` functions (but no other forms of `inline`, for which GCC and C99 have differing interpretations).
- `long long`

- `bool` and `<stdbool.h>`, but don't assume that `bool` or `_Bool` can only take on the values `0` or `1`, because this behavior can't be simulated on C89 compilers.

  Also, don't assume that a conversion to `bool` or `_Bool` follows C99 semantics, i.e. use `(bool) (some_value != 0)` rather than `(bool) some_value`. The latter might produce unexpected results on non-C99 environments. For example, if `bool` is implemented as a typedef of char and `some_value = 0x10000000`.

- Designated initializers (e.g. `struct foo foo = { .a = 1 };` and `int a[] = { [2] = 5 };`).
- Mixing of declarations and code within a block. Favor positioning that allows variables to be initialized at their point of declaration.
- Use of declarations in iteration statements (e.g. `for (int i = 0; i < 10; i++)`).
- Use of a trailing comma in an enum declaration (e.g. `enum { x = 1, };`).

As a matter of style, avoid `//` comments.

Avoid using GCC or Clang extensions unless you also add a fallback for other compilers. You can, however, use C99 features or GCC extensions also supported by Clang in code that compiles only on GNU/Linux (such as `lib/netdev-linux.c`), because GCC is the system compiler there.

Python

When introducing new Python code, try to follow Python's PEP 8 style. Consider running the `pep8` or `flake8` tool against your code to find issues.

Libraries

When introducing a new library, follow *Open vSwitch Library ABI guide*

8.1.4 OVN Documentation Style

This file describes the documentation style used in all documentation found in OVN. Documentation includes any documents found in `Documentation` along with any `README`, `MAINTAINERS`, or generally `rst` suffixed documents found in the project tree.

reStructuredText vs. Sphinx

reStructuredText (rST) is the syntax, while Sphinx is a documentation generator. Sphinx introduces a number of extensions to rST, like the `:ref:` role, which can and should be used in documentation, but these will not work correctly on GitHub. As such, these extensions should not be used in any documentation in the root level, such as the `README`.

rST Conventions

Basics

Many of the basic documentation guidelines match those of the *OVN Coding Style*.

- Use reStructuredText (rST) for all documentation.

  Sphinx extensions can be used, but only for documentation in the `Documentation` folder.

- Limit lines at 79 characters.

  Note: An exception to this rule is text within code-block elements that cannot be wrapped and links within references.

- Use spaces for indentation.
- Match indentation levels.

  A change in indentation level usually signifies a change in content nesting, by either closing the existing level or introducing a new level.

- Avoid trailing spaces on lines.
- Include a license (see this file) in all docs.
- Most importantly, always build and display documentation before submitting changes! Docs aren't unit testable, so visible inspection is necessary.

File Names

- Use hyphens as space delimiters. For example: `my-readme-document.rst`

  Note: An exception to this rule is any man pages, which take an trailing number corresponding to the number of arguments required. This number is preceded by an underscore.

- Use lowercase filenames.

  Note: An exception to this rule is any documents found in the root-level of the project.

Titles

- Use the following headers levels.

  ======= Heading 0 (reserved for the title in a document)
  ------- Heading 1
  ~~~~~~~ Heading 2
  +++++++ Heading 3
  ''''''' Heading 4

  Note: Avoid using lower heading levels by rewriting and reorganizing the information.

- Under- and overlines should be of the same length as that of the heading text.
- Use "title case" for headers.
  ~~~~~~~

Code

- Use `::` to prefix code.
- Don't use syntax highlighting such as `.. highlight:: <syntax>` or `code-block:: <syntax>` because it depends on external `pygments` library.
- Prefix commands with `$`.
- Where possible, include fully-working snippets of code. If there pre-requisites, explain what they are and how to achieve them.

Admonitions

- Use admonitions to call attention to important information.:

```
.. note::

   This is a sample callout for some useful tip or trick.
```

  Example admonitions include: `warning`, `important`, `note`, `tip` or `seealso`.

- Use notes sparingly. Avoid having more than one per subsection.

Tables

- Use either graphic tables, list tables or CSV tables.

Graphic tables

```
.. table:: OVS-Linux kernel compatibility

  ============ ==============
  Open vSwitch Linux kernel
  ============ ==============
  1.4.x        2.6.18 to 3.2
  1.5.x        2.6.18 to 3.2
  1.6.x        2.6.18 to 3.2
  ============ ==============
```

```
.. table:: OVS-Linux kernel compatibility

  +--------------+---------------+
  | Open vSwitch | Linux kernel  |
  +==============+===============+
  | 1.4.x        | 2.6.18 to 3.2 |
  +--------------+---------------+
  | 1.5.x        | 2.6.18 to 3.2 |
  +--------------+---------------+
  | 1.6.x        | 2.6.18 to 3.2 |
  +--------------+---------------+
```

Note: The `table` role - `.. table:: <name>` - can be safely omitted.

List tables

```
.. list-table:: OVS-Linux kernel compatibility
   :widths: 10 15
   :header-rows: 1

   * - Open vSwitch
     - Linux kernel
   * - 1.4.x
     - 2.6.18 to 3.2
   * - 1.5.x
     - 2.6.18 to 3.2
   * - 1.6.x
     - 2.6.18 to 3.2
```

CSV tables

```
.. csv-table:: OVS-Linux kernel compatibility
   :header: Open vSwitch, Linux kernel
   :widths: 10 15

   1.4.x, 2.6.18 to 3.2
   1.5.x, 2.6.18 to 3.2
   1.6.x, 2.6.18 to 3.2
```

Cross-referencing

- To link to an external file or document, include as a link.:

```
Here's a `link <http://openvswitch.org>`__ to the Open vSwitch website.


Here's a `link`_ in reference style.

.. _link: http://openvswitch.org
```

- You can also use citations.:

```
Refer to the Open vSwitch documentation [1]_.

References
----------

.. [1]: http://openvswitch.org
```

- To cross-reference another doc, use the `doc` role.:

```
Here is a link to the :doc:`/README.rst`
```

Note: This is a Sphinx extension. Do not use this in any top-level documents.

- To cross-reference an arbitrary location in a doc, use the `ref` role.:

```
.. _sample-crossref

Title
~~~~~

Hello, world.

Another Title
~~~~~~~~~~~~~

Here is a cross-reference to :ref:`sample-crossref`.
```

Note: This is a Sphinx extension. Do not use this in any top-level documents.

Figures and Other Media

- All images should be in PNG format and compressed where possible. For PNG files, use OptiPNG and AdvanceCOMP's `advpng`:

```
$ optipng -o7 -zm1-9 -i0 -strip all <path_to_png>
$ advpng -z4 <path_to_png>
```

- Any ASCII text "images" should be included in code-blocks to preserve formatting
- Include other reStructuredText verbatim in a current document

Comments

- Comments are indicated by means of the `..` marker.:

```
.. TODO(stephenfin) This section needs some work. This TODO will not
   appear in the final generated document, however.
```

Man Pages

In addition to the above, man pages have some specific requirements:

- You **must** define the following sections:
  - Synopsis
  - Description
  - Options

Note that *NAME* is not included - this is automatically generated by Sphinx and should not be manually defined. Also note that these do not need to be uppercase - Sphinx will do this automatically.

Additional sections are allowed. Refer to *man-pages(8)* for information on the sections generally allowed.

- You **must not** define a *NAME* section.

  See above.

- The *OPTIONS* section must describe arguments and options using the program and option directives.

  This ensures the output is formatted correctly and that you can cross-reference various programs and commands from the documentation. For example:

```
.. program:: ovs-do-something

.. option:: -f, --force

    Force the operation

.. option:: -b <bridge>, --bridge <bridge>

    Name or ID of bridge
```

Important: Option argument names should be enclosed in angle brackets, as above.

- Any references to the application or any other OVN application must be marked up using the *program* role.

  This allows for easy linking in the HTML output and correct formatting in the man page output. For example:

```
To do something, run :program:`ovs-do-something`.
```

- The man page must be included in the list of man page documents found in conf.py

Refer to existing man pages, such as *ovn-sim* for a worked example.

Writing Style

Follow these guidelines to ensure readability and consistency of the Open vSwitch documentation. These guidelines are based on the /*IBM Style Guide/*.

- Use standard US English

  Use a spelling and grammar checking tool as necessary.

- Expand initialisms and acronyms on first usage.

  Commonly used terms like CPU or RAM are allowed.

Do not use	Do use
OVS is a virtual switch. OVS has…	Open vSwitch (OVS) is a virtual switch. OVS has…
The VTEP emulator is…	The Virtual Tunnel Endpoint (VTEP) emulator is…

- Write in the active voice

  The subject should do the verb's action, rather than be acted upon.

Do not use	Do use
A bridge is created by you	Create a bridge

- Write in the present tense

Do not use	Do use
Once the bridge is created, you can create a port	Once the bridge is created, create a port

- Write in second person

Do not use	Do use
To create a bridge, the user runs:	To create a bridge, run:

- Keep sentences short and concise
- Eliminate needless politeness

  Avoid "please" and "thank you"

Helpful Tools

There are a number of tools, online and offline, which can be used to preview documents are you edit them:

- rst.ninjs.org

  An online rST editor/previewer

- ReText

  A simple but powerful editor for Markdown and reStructuredText. ReText is written in Python.

- restview

  A viewer for ReStructuredText documents that renders them on the fly.

Useful Links

- Quick reStructuredText
- Sphinx Documentation

8.1.5 Open vSwitch Library ABI Updates

This file describes the manner in which the Open vSwitch shared library manages different ABI and API revisions. This document aims to describe the background, goals, and concrete mechanisms used to export code-space functionality so that it may be shared between multiple applications.

Definitions

Table 1: Definitions for terms appearing in this document

Term	Definition
ABI	Abbreviation of Application Binary Interface
API	Abbreviation of Application Programming Interface
Application Binary Interface	The low-level runtime interface exposed by an object file.
Application Programming Interface	The source-code interface descriptions intended for use in multiple translation units when compiling.
Code library	A collection of function implementations and definitions intended to be exported and called through a well-defined interface.
Shared Library	A code library which is imported at run time.

Overview

C and C++ applications often use 'external' functionality, such as printing specialized data types or parsing messages, which has been exported for common use. There are many possible ways for applications to call such external functionality, for instance by including an appropriate inline definition which the compiler can emit as code in each function it appears. One such way of exporting and importing such functionality is through the use of a library of code.

When a compiler builds object code from source files to produce object code, the results are binary data arranged with specific calling conventions, alignments, and order suitable for a run-time environment or linker. This result defines a specific ABI.

As library of code develops and its exported interfaces change over time, the resulting ABI may change as well. Therefore, care must be taken to ensure the changes made to libraries of code are effectively communicated to applications which use them. This includes informing the applications when incompatible changes are made.

The Open vSwitch project exports much of its functionality through multiple such libraries of code. These libraries are intended for multiple applications to import and use. As the Open vSwitch project continues to evolve and change, its exported code will evolve as well. To ensure that applications linking to these libraries are aware of these changes, Open vSwitch employs libtool version stamps.

ABI Policy

Open vSwitch will export the ABI version at the time of release, such that the library name will be the major.minor version, and the rest of the release version information will be conveyed with a libtool interface version.

The intent is for Open vSwitch to maintain an ABI stability for each minor revision only (so that Open vSwitch release 2.5 carries a guarantee for all 2.5.ZZ micro-releases). This means that any porting effort to stable branches must take not to disrupt the existing ABI.

In the event that a bug must be fixed in a backwards-incompatible way, developers must bump the libtool 'current' version to inform the linker of the ABI breakage. This will signal that libraries exposed by the subsequent release will not maintain ABI stability with the previous version.

Coding

At build time, if building shared libraries by passing the *–enable-shared* arguments to *./configure*, version information is extracted from the `$PACKAGE_VERSION` automake variable and formatted into the appropriate arguments. These get exported for use in Makefiles as `$OVS_LTINFO`, and passed to each exported library along with other `LDFLAGS`.

Therefore, when adding a new library to the build system, these version flags should be included with the `$LDFLAGS` variable. Nothing else needs to be done.

Changing an exported function definition (from a file in, for instance *lib/*.h*) is only permitted from minor release to minor release. Likewise changes to library data structures should only occur from minor release to minor release.

8.2 Mailing Lists

Important: Report security issues **only** to security@openvswitch.org. For more information, refer to our *security policies*.

8.2.1 ovs-announce

The ovs-announce mailing list is used to announce new versions of Open vSwitch and OVN and is extremely low-volume. (subscribe) (archives)

8.2.2 ovs-discuss

The ovs-discuss mailing list is used to discuss plans and design decisions for Open vSwitch and OVN. It is also an appropriate place for user questions. (subscribe) (archives)

8.2.3 ovs-dev

The ovs-dev mailing list is used to discuss development and review code before being committed. (subscribe) (archives)

8.2.4 bugs

The bugs mailing list is an alias for the discuss mailing list.

8.2.5 security

The security mailing list is for submitting security vulnerabilities to the security team.

8.3 Patchwork

Open vSwitch and OVN use Patchwork to track the status of patches sent to the *ovs-dev mailing list*. Our Patchwork instance can be found on ozlabs.org.

Patchwork provides a number of useful features for developers working on Open vSwitch and OVN:

- Tracking the lifecycle of patches (accepted, rejected, under-review, . . .)
- Assigning reviewers (delegates) to patches
- Downloading/applying patches, series, and bundles via the web UI or the REST API (see *git-pw*)
- A usable UI for viewing patch discussions

8.3.1 git-pw

The *git-pw* tool provides a way to download and apply patches, series, and bundles. You can install *git-pw* from PyPi like so:

```
$ pip install --user git-pw
```

To actually use *git-pw*, you must configure it with the Patchwork instance URL, Patchwork project, and your Patchwork user authentication token. The URL and project are provided below, but you must obtain your authentication token from your Patchwork User Profile page. If you do not already have a Patchwork user account, you should create one now.

Once your token is obtained, configure *git-pw* as below. Note that this must be run from within the OVN Git repository:

```
$ git config pw.server https://patchwork.ozlabs.org/
$ git config pw.project ovn
$ git config pw.token $PW_TOKEN  # using the token obtained earlier
```

Once configured, run the following to get information about available commands:

```
$ git pw --help
```

8.3.2 pwclient

The *pwclient* is a legacy tool that provides some of the functionality of *git-pw* but uses the legacy XML-RPC API. It is considered deprecated in its current form and *git-pw* should be used instead.

8.4 OVN Release Process

This document describes the process ordinarily used for OVN development and release. Exceptions are sometimes necessary, so all of the statements here should be taken as subject to change through rough consensus of OVN contributors, obtained through public discussion on, e.g., ovs-dev or the #openvswitch IRC channel.

8.4.1 Release Strategy

OVN feature development takes place on the "master" branch. Ordinarily, new features are rebased against master and applied directly. For features that take significant development, sometimes it is more appropriate to merge a separate branch into master; please discuss this on ovs-dev in advance.

The process of making a release has the following stages. See *Release Scheduling* for the timing of each stage:

1. "Soft freeze" of the master branch.

   During the freeze, we ask committers to refrain from applying patches that add new features unless those patches were already being publicly discussed and reviewed before the freeze began. Bug fixes are welcome at any time. Please propose and discuss exceptions on ovs-dev.

2. Fork a release branch from master, named for the expected release number, e.g. "branch-2019.10" for the branch that will yield OVN 2019.10.x.

   Release branches are intended for testing and stabilization. At this stage and in later stages, they should receive only bug fixes, not new features. Bug fixes applied to release branches should be backports of corresponding bug fixes to the master branch, except for bugs present only on release branches (which are rare in practice).

At this stage, sometimes there can be exceptions to the rule that a release branch receives only bug fixes. Like bug fixes, new features on release branches should be backports of the corresponding commits on the master branch. Features to be added to release branches should be limited in scope and risk and discussed on ovs-dev before creating the branch.

3. When committers come to rough consensus that the release is ready, they release the .0 release on its branch, e.g. 2019.10.0 for branch-2019.10. To make the actual release, a committer pushes a signed tag named, e.g. v2019.10.0, to the OVN repository, makes a release tarball available on openvswitch.org, and posts a release announcement to ovs-announce.
4. As bug fixes accumulate, or after important bugs or vulnerabilities are fixed, committers may make additional releases from a branch: 2019.10.1, 2019.10.2, and so on. The process is the same for these additional release as for a .0 release.

8.4.2 Long-term Support Releases

The OVN project will periodically designate a release as "long-term support" or LTS for short. An LTS release has the distinction of being maintained for longer than a standard release.

LTS releases will receive bug fixes until the point that another LTS is released. At that point, the old LTS will receive an additional year of critical and security fixes. Critical fixes are those that are required to ensure basic operation (e.g. memory leak fixes, crash fixes). Security fixes are those that address concerns about exploitable flaws in OVN and that have a corresponding CVE report.

LTS releases are scheduled to be released once every two years. This means that any given LTS will receive bug fix support for two years, followed by one year of critical bug fixes and security fixes.

8.4.3 Release Numbering

The version number on master should normally end in .90. This indicates that the OVN version is "almost" the next version to branch.

Forking master into branch-x.y requires two commits to master. The first is titled "Prepare for x.y.0" and increments the version number to x.y. This is the initial commit on branch-x.y. The second is titled "Prepare for post-x.y.0 (x.y.90)" and increments the version number to x.y.90.

The version number on a release branch is x.y.z, where x is the current year, y is the month of the release, and z is initially 0. Making a release requires two commits. The first is titled *Set release dates for x.y.z.* and updates NEWS and debian/changelog to specify the release date of the new release. This commit is the one made into a tarball and tagged. The second is titled *Prepare for x.y.(z+1).* and increments the version number and adds a blank item to NEWS with an unspecified date.

8.4.4 Release Scheduling

OVN makes releases at the following three-month cadence. All dates are approximate:

Time (months)	Example Dates	Stage
T	Dec 1, Mar 1, . . .	Begin x.y release cycle
T + 2	Feb 1, May 1, . . .	"Soft freeze" master for x.y release
T + 2.5	Feb 15, May 15, . . .	Fork branch-x.y from master
T + 3	Mar 1, Jun 1, . . .	Release version x.y.0

8.4.5 Release Calendar

The 2021 timetable is shown below. Note that these dates are not set in stone. If extenuating circumstances arise, a release may be delayed from its target date.

Release	Soft Freeze	Branch Creation	Release
21.03.0	Feb 5	Feb 19	Mar 5
21.06.0	May 7	May 21	Jun 4
21.09.0	Aug 6	Aug 20	Sep 3
21.12.0	Nov 5	Nov 19	Dec 3

Below is the 2022 timetable

Release	Soft Freeze	Branch Creation	Release
22.03.0	Feb 4	Feb 18	Mar 4
22.06.0	May 6	May 20	Jun 3
22.09.0	Aug 5	Aug 19	Sep 2
22.12.0	Nov 4	Nov 18	Dec 2

8.4.6 Contact

Use dev@openvswitch.org to discuss the OVN development and release process.

8.5 Reporting Bugs in OVN

We are eager to hear from users about problems that they have encountered with OVN. This file documents how best to report bugs so as to ensure that they can be fixed as quickly as possible.

Please report bugs by sending email to bugs@openvswitch.org.

For reporting security vulnerabilities, please read *OVN's Security Process*.

The most important parts of your bug report are the following:

- What you did that make the problem appear.
- What you expected to happen.
- What actually happened.

Please also include the following information:

- The OVN version number (as output by `ovn-controller --version`).
- The Git commit number (as output by `git rev-parse HEAD`), if you built from a Git snapshot.
- Any local patches or changes you have applied (if any).

The following are also handy sometimes:

- The kernel version on which Open vSwitch is running (from `/proc/version`) and the distribution and version number of your OS (e.g. "Centos 5.0").
- The contents of the northbound database.
- Any other information that you think might be relevant.

Important: bugs@openvswitch.org is a public mailing list, to which anyone can subscribe, so do not include confidential information in your bug report.

8.6 OVN's Security Process

This is a proposed security vulnerability reporting and handling process for OVN. It is based on the OpenStack vulnerability management process described at https://wiki.openstack.org/wiki/Vulnerability_Management.

The OVN security team coordinates vulnerability management using the ovs-security mailing list. Membership in the security team and subscription to its mailing list consists of a small number of trustworthy people, as determined by rough consensus of the OVN committers. The OVN security team should include OVN committers, to ensure prompt and accurate vulnerability assessments and patch review.

We encourage everyone involved in the security process to GPG-sign their emails. We additionally encourage GPG-encrypting one-on-one conversations as part of the security process.

8.6.1 What is a vulnerability?

All vulnerabilities are bugs, but not every bug is a vulnerability. Vulnerabilities compromise one or more of:

- Confidentiality (personal or corporate confidential data).
- Integrity (trustworthiness and correctness).
- Availability (uptime and service).

Here are some examples of vulnerabilities to which one would expect to apply this process:

- A crafted packet that causes a kernel or userspace crash (Availability).
- A flow translation bug that misforwards traffic in a way likely to hop over security boundaries (Integrity).
- An OpenFlow protocol bug that allows a controller to read arbitrary files from the file system (Confidentiality).
- Misuse of the OpenSSL library that allows bypassing certificate checks (Integrity).
- A bug (memory corruption, overflow, ...) that allows one to modify the behaviour of OVN through external configuration interfaces such as OVSDB (Integrity).
- Privileged information is exposed to unprivileged users (Confidentiality).

If in doubt, please do use the vulnerability management process. At worst, the response will be to report the bug through the usual channels.

8.6.2 Step 1: Reception

To report an OVN vulnerability, send an email to the ovs-security mailing list (see *contact* at the end of this document). A security team member should reply to the reporter acknowledging that the report has been received.

Consider reporting the information mentioned in *Reporting Bugs in OVN*, where relevant.

Reporters may ask for a GPG key while initiating contact with the security team to deliver more sensitive reports.

8.6.3 Step 2: Assessment

The security team should discuss the vulnerability. The reporter should be included in the discussion (via "CC") to an appropriate degree.

The assessment should determine which OVN versions are affected (e.g. every version, only the latest release, only unreleased versions), the privilege required to take advantage of the vulnerability (e.g. any network user, any local L2 network user, any local system user, connected OpenFlow controllers), the severity of the vulnerability, and how the vulnerability may be mitigated (e.g. by disabling a feature).

The treatment of the vulnerability could end here if the team determines that it is not a realistic vulnerability.

8.6.4 Step 3a: Document

The security team develops a security advisory document. The security team may, at its discretion, include the reporter (via "CC") in developing the security advisory document, but in any case should accept feedback from the reporter before finalizing the document. When the document is final, the security team should obtain a CVE for the vulnerability from a CNA (https://cve.mitre.org/cve/cna.html).

The document credits the reporter and describes the vulnerability, including all of the relevant information from the assessment in step 2. Suitable sections for the document include:

```
* Title: The CVE identifier, a short description of the
  vulnerability.  The title should mention OVN.

  In email, the title becomes the subject.  Pre-release advisories
  are often passed around in encrypted email, which have plaintext
  subjects, so the title should not be too specific.

* Description: A few paragraphs describing the general
  characteristics of the vulnerability, including the versions of
  OVN that are vulnerable, the kind of attack that
  exposes the vulnerability, and potential consequences of the
  attack.

  The description should re-state the CVE identifier, in case the
  subject is lost when an advisory is sent over email.

* Mitigation: How an OVN administrator can minimize the
  potential for exploitation of the vulnerability, before applying
  a fix.  If no mitigation is possible or recommended, explain
  why, to reduce the chance that at-risk users believe they are
  not at risk.

* Fix: Describe how to fix the vulnerability, perhaps in terms of
  applying a source patch.  The patch or patches themselves, if
  included in the email, should be at the very end of the advisory
  to reduce the risk that a reader would stop reading at this
  point.

* Recommendation: A concise description of the security team's
  recommendation to users.

* Acknowledgments: Thank the reporters.

* Vulnerability Check: A step-by-step procedure by which a user
  can determine whether an installed copy of OVN is vulnerable.
```

(continues on next page)

(continued from previous page)

```
  The procedure should clearly describe how to interpret the
  results, including expected results in vulnerable and
  not-vulnerable cases.  Thus, procedures that produce clear and
  easily distinguished results are preferred.

  The procedure should assume as little understanding of Open
  vSwitch as possible, to make it more likely that a competent
  administrator who does not specialize in OVN can
  perform it successfully.

  The procedure should have minimal dependencies on tools that are
  not widely installed.

  Given a choice, the procedure should be one that takes at least
  some work to turn into a useful exploit.  For example, a
  procedure based on "ovs-appctl" commands, which require local
  administrator access, is preferred to one that sends test
  packets to a machine, which only requires network connectivity.

  The section should say which operating systems it is designed
  for.  If the procedure is likely to be specific to particular
  architectures (e.g. x86-64, i386), it should state on which ones
  it has been tested.

  This section should state the risks of the procedure.  For
  example, if it can crash OVN or disrupt packet forwarding, say so.

  It is more useful to explain how to check an installed and
  running OVN than one built locally from source, but if
  it is easy to use the procedure from a sandbox environment, it
  can be helpful to explain how to do so.

* Patch: If a patch or patches are available, and it is practical
  to include them in the email, put them at the end.  Format them
  as described in :doc:`contributing/submitting-patches`, that is, as
  output by "git format-patch".

  The patch subjects should include the version for which they are
  suited, e.g. "[PATCH branch-2.3]" for a patch against OVN 2.3.x.
  If there are multiple patches for multiple versions of OVN, put
  them in separate sections with clear titles.

  Multiple patches for a single version of OVN, that must
  be stacked on top of each other to fix a single vulnerability,
  are undesirable because users are less likely to apply all of
  them correctly and in the correct order.

  Each patch should include a Vulnerability tag with the CVE
  identifier, a Reported-by tag or tags to credit the reporters,
  and a Signed-off-by tag to acknowledge the Developer's
  Certificate of Origin.  It should also include other appropriate
  tags, such as Acked-by tags obtained during review.
```

CVE-2016-2074 is an example advisory document.

8.6.5 Step 3b: Fix

Steps 3a and 3b may proceed in parallel.

The security team develops and obtains (private) reviews for patches that fix the vulnerability. If necessary, the security team pulls in additional developers, who must agree to maintain confidentiality.

8.6.6 Step 4: Embargoed Disclosure

The security advisory and patches are sent to downstream stakeholders, with an embargo date and time set from the time sent. Downstream stakeholders are expected not to deploy or disclose patches until the embargo is passed.

A disclosure date is negotiated by the security team working with the bug submitter as well as vendors. However, the OVN security team holds the final say when setting a disclosure date. The timeframe for disclosure is from immediate (esp. if it's already publicly known) to a few weeks. As a basic default policy, we expect report date to disclosure date to be 10 to 15 business days.

Operating system vendors are obvious downstream stakeholders. It may not be necessary to be too choosy about who to include: any major OVN user who is interested and can be considered trustworthy enough could be included. To become a downstream stakeholder, email the ovs-security mailing list.

If the vulnerability is already public, skip this step.

8.6.7 Step 5: Public Disclosure

When the embargo expires, push the (reviewed) patches to appropriate branches, post the patches to the ovs-dev mailing list (noting that they have already been reviewed and applied), post the security advisory to appropriate mailing lists (ovs-announce, ovs-discuss), and post the security advisory on the OVN webpage.

When the patch is applied to LTS (long-term support) branches, a new version should be released.

The security advisory should be GPG-signed by a security team member with a key that is in a public web of trust.

Contact

Report security vulnerabilities to the ovs-security mailing list: security@openvswitch.org

Report problems with this document to the ovs-bugs mailing list: bugs@openvswitch.org

8.7 Emeritus Status for OVN Committers

OVN committers are nominated and elected based on their impact on the OVN project. Over time, as committers' responsibilities change, some may become unable or uninterested in actively participating in project governance. Committer "emeritus" status provides a way for committers to take a leave of absence from OVN governance responsibilities. The following guidelines clarify the process around the emeritus status for committers:

- A committer may choose to transition from active to emeritus, or from emeritus to active, by sending an email to the committers mailing list.
- If a committer hasn't been heard from in 6 months, and does not respond to reasonable attempts to contact him or her, the other committers can vote as a majority to transition the committer from active to emeritus. (If the committer resurfaces, he or she can transition back to active by sending an email to the committers mailing list.)
- Emeritus committers may stay on the committers mailing list to continue to follow any discussions there.

- Emeritus committers do not nominate or vote in committer elections. From a governance perspective, they are equivalent to a non-committer.
- Emeritus committers cannot merge patches to the OVN repository.
- Emeritus committers will be listed in a separate section in the MAINTAINERS.rst file to continue to recognize their contributions to the project.

Emeritus status does not replace the procedures for forcibly removing a committer.

Note that just because a committer is not able to work on the project on a day-to-day basis, we feel they are still capable of providing input on the direction of the project. No committer should feel pressured to move themselves to this status. Again, it's just an option for those that do not currently have the time or interest.

8.8 Expectations for Developers with OVN Repo Access

8.8.1 Pre-requisites

Be familiar with the guidelines and standards defined in *Contributing to OVN*.

8.8.2 Review

Code (yours or others') must be reviewed publicly (by you or others) before you push it to the repository. With one exception (see below), every change needs at least one review.

If one or more people know an area of code particularly well, code that affects that area should ordinarily get a review from one of them.

The riskier, more subtle, or more complicated the change, the more careful the review required. When a change needs careful review, use good judgment regarding the quality of reviews. If a change adds 1000 lines of new code, and a review posted 5 minutes later says just "Looks good," then this is probably not a quality review.

(The size of a change is correlated with the amount of care needed in review, but it is not strictly tied to it. A search and replace across many files may not need much review, but one-line optimization changes can have widespread implications.)

Your own small changes to fix a recently broken build ("make") or tests ("make check"), that you believe to be visible to a large number of developers, may be checked in without review. If you are not sure, ask for review. If you do push a build fix without review, send the patch to ovs-dev afterward as usual, indicating in the email that you have already pushed it.

Regularly review submitted code in areas where you have expertise. Consider reviewing other code as well.

8.8.3 Git conventions

Do not push merge commits to the Git repository without prior discussion on ovs-dev.

If you apply a change (yours or another's) then it is your responsibility to handle any resulting problems, especially broken builds and other regressions. If it is someone else's change, then you can ask the original submitter to address it. Regardless, you need to ensure that the problem is fixed in a timely way. The definition of "timely" depends on the severity of the problem.

If a bug is present on master and other branches, fix it on master first, then backport the fix to other branches. Straightforward backports do not require additional review (beyond that for the fix on master).

Feature development should be done only on master. Occasionally it makes sense to add a feature to the most recent release branch, before the first actual release of that branch. These should be handled in the same way as bug fixes, that is, first implemented on master and then backported.

Keep the authorship of a commit clear by maintaining a correct list of "Signed-off-by:"s. If a confusing situation comes up, as it occasionally does, bring it up on the mailing list. If you explain the use of "Signed-off-by:" to a new developer, explain not just how but why, since the intended meaning of "Signed-off-by:" is more important than the syntax. As part of your explanation, quote or provide a URL to the Developer's Certificate of Origin in *Submitting Patches*.

Use Reported-by: and Tested-by: tags in commit messages to indicate the source of a bug report.

Keep the `AUTHORS.rst` file up to date.

8.8.4 Pre-Push Hook

The following script can be helpful because it provides an extra chance to check for mistakes while pushing to the main branch. If you would like to use it, install it as `hooks/pre-push` in your `.git` directory and make sure to mark it as executable with `chmod +x`. For maximum utility, make sure `checkpatch.py` is in `$PATH`:

```
#! /bin/bash

remote=$1

case $remote in
    ovs|ovn|origin) ;;
    *) exit 0 ;;
esac

while read local_ref local_sha1 remote_ref remote_sha1; do
    case $remote_ref in
        refs/heads/main)
            n=0
            while read sha
            do
                n=$(expr $n + 1)
                git log -1 $sha
                echo
                checkpatch.py -1 $sha
            done <<EOF
$(git --no-pager log --pretty=%H $local_sha1...$remote_sha1)
EOF

            b=${remote_ref#refs/heads/}
            echo "You're about to push $n commits to protected branch $b on $remote."

            read -p "Do you want to proceed? [y|n] " reply < /dev/tty
            if echo $reply | grep -E '^[Yy]$' > /dev/null; then
                :
            else
                exit 1
            fi
            ;;
    esac
done

exit 0
```

8.9 OVN Committer Grant/Revocation Policy

An OVN committer is a participant in the project with the ability to commit code directly to the master repository. Commit access grants a broad ability to affect the progress of the project as presented by its most important artifact, the code and related resources that produce working binaries of OVN As such it represents a significant level of trust in an individual's commitment to working with other committers and the community at large for the benefit of the project. It can not be granted lightly and, in the worst case, must be revocable if the trust placed in an individual was inappropriate.

This document suggests guidelines for granting and revoking commit access. It is intended to provide a framework for evaluation of such decisions without specifying deterministic rules that wouldn't be sensitive to the nuance of specific situations. In the end the decision to grant or revoke committer privileges is a judgment call made by the existing set of committers.

8.9.1 Granting Commit Access

Granting commit access should be considered when a candidate has demonstrated the following in their interaction with the project:

- Contribution of significant new features through the patch submission process where:
  - Submissions are free of obvious critical defects
  - Submissions do not typically require many iterations of improvement to be accepted
- Consistent participation in code review of other's patches, including existing committers, with comments consistent with the overall project standards
- Assistance to those in the community who are less knowledgeable through active participation in project forums such as the ovs-discuss mailing list.
- Plans for sustained contribution to the project compatible with the project's direction as viewed by current committers.
- Commitment to meet the expectations described in the "Expectations of Developer's with OVN Access"

The process to grant commit access to a candidate is simple:

- An existing committer nominates the candidate by sending an email to all existing committers with information substantiating the contributions of the candidate in the areas described above.
- All existing committers discuss the pros and cons of granting commit access to the candidate in the email thread.
- When the discussion has converged or a reasonable time has elapsed without discussion developing (e.g. a few business days) the nominator calls for a final decision on the candidate with a followup email to the thread.
- Each committer may vote yes, no, or abstain by replying to the email thread. A failure to reply is an implicit abstention.
- After votes from all existing committers have been collected or a reasonable time has elapsed for them to be provided (e.g. a couple of business days) the votes are evaluated. To be granted commit access the candidate must receive yes votes from a majority of the existing committers and zero no votes. Since a no vote is effectively a veto of the candidate it should be accompanied by a reason for the vote.
- The nominator summarizes the result of the vote in an email to all existing committers.
- If the vote to grant commit access passed, the candidate is contacted with an invitation to become a committer to the project which asks them to agree to the committer expectations documented on the project web site.
- If the candidate agrees access is granted by setting up commit access to the repos on github.

8.9.2 Revoking Commit Access

When a committer behaves in a manner that other committers view as detrimental to the future of the project, it raises a delicate situation with the potential for the creation of division within the greater community. These situations should be handled with care. The process in this case is:

- Discuss the behavior of concern with the individual privately and explain why you believe it is detrimental to the project. Stick to the facts and keep the email professional. Avoid personal attacks and the temptation to hypothesize about unknowable information such as the other's motivations. Make it clear that you would prefer not to discuss the behavior more widely but will have to raise it with other contributors if it does not change. Ideally the behavior is eliminated and no further action is required. If not,
- Start an email thread with all committers, including the source of the behavior, describing the behavior and the reason it is detrimental to the project. The message should have the same tone as the private discussion and should generally repeat the same points covered in that discussion. The person whose behavior is being questioned should not be surprised by anything presented in this discussion. Ideally the wider discussion provides more perspective to all participants and the issue is resolved. If not,
- Start an email thread with all committers except the source of the detrimental behavior requesting a vote on revocation of commit rights. Cite the discussion among all committers and describe all the reasons why it was not resolved satisfactorily. This email should be carefully written with the knowledge that the reasoning it contains may be published to the larger community to justify the decision.
- Each committer may vote yes, no, or abstain by replying to the email thread. A failure to reply is an implicit abstention.
- After all votes have been collected or a reasonable time has elapsed for them to be provided (e.g. a couple of business days) the votes are evaluated. For the request to revoke commit access for the candidate to pass it must receive yes votes from two thirds of the existing committers.
- anyone that votes no must provide their reasoning, and
- if the proposal passes then counter-arguments for the reasoning in no votes should also be documented along with the initial reasons the revocation was proposed. Ideally there should be no new counter-arguments supplied in a no vote as all concerns should have surfaced in the discussion before the vote.
- The original person to propose revocation summarizes the result of the vote in an email to all existing committers excepting the candidate for removal.
- If the vote to revoke commit access passes, access is removed and the candidate for revocation is informed of that fact and the reasons for it as documented in the email requesting the revocation vote.
- Ideally the revoked committer peacefully leaves the community and no further action is required. However, there is a distinct possibility that he/she will try to generate support for his/her point of view within the larger community. In this case the reasoning for removing commit access as described in the request for a vote will be published to the community.

8.9.3 Changing the Policy

The process for changing the policy is:

- Propose the changes to the policy in an email to all current committers and request discussion.
- After an appropriate period of discussion (a few days) update the proposal based on feedback if required and resend it to all current committers with a request for a formal vote.
- After all votes have been collected or a reasonable time has elapsed for them to be provided (e.g. a couple of business days) the votes are evaluated. For the request to modify the policy to pass it must receive yes votes from two thirds of the existing committers.

Template Emails

8.9.4 Nomination to Grant Commit Access

I would like to nominate *[candidate]* for commit access. I believe *[he/she]* has met the conditions for commit access described in the committer grant policy on the project web site in the following ways:

[list of requirements & evidence]

Please reply to all in this message thread with your comments and questions. If that discussion concludes favorably I will request a formal vote on the nomination in a few days.

8.9.5 Vote to Grant Commit Access

I nominated *[candidate]* for commit access on *[date]*. Having allowed sufficient time for discussion it's now time to formally vote on the proposal.

Please reply to all in this thread with your vote of: YES, NO, or ABSTAIN. A failure to reply will be counted as an abstention. If you vote NO, by our policy you must include the reasons for that vote in your reply. The deadline for votes is *[date and time]*.

If a majority of committers vote YES and there are zero NO votes commit access will be granted.

8.9.6 Vote Results for Grant of Commit Access

The voting period for granting to commit access to *[candidate]* initiated at *[date and time]* is now closed with the following results:

YES: *[count of yes votes]* (*[% of voters]*)

NO: *[count of no votes]* (*[% of voters]*)

ABSTAIN: *[count of abstentions]* (*[% of voters]*)

Based on these results commit access *[is/is NOT]* granted.

8.9.7 Invitation to Accepted Committer

Due to your sustained contributions to the OVN project we would like to provide you with commit access to the project repository. Developers with commit access must agree to fulfill specific responsibilities described in the source repository:

/Documentation/internals/committer-responsibilities.rst

Please let us know if you would like to accept commit access and if so that you agree to fulfill these responsibilities. Once we receive your response we'll set up access. We're looking forward continuing to work together to advance the OVN project.

8.9.8 Proposal to Revoke Commit Access for Detrimental Behavior

I regret that I feel compelled to propose revocation of commit access for *[candidate]*. I have privately discussed with *[him/her]* the following reasons I believe *[his/her]* actions are detrimental to the project and we have failed to come to a mutual understanding:

[List of reasons and supporting evidence]

Please reply to all in this thread with your thoughts on this proposal. I plan to formally propose a vote on the proposal on or after *[date and time]*.

It is important to get all discussion points both for and against the proposal on the table during the discussion period prior to the vote. Please make it a high priority to respond to this proposal with your thoughts.

8.9.9 Vote to Revoke Commit Access

I nominated *[candidate]* for revocation of commit access on *[date]*. Having allowed sufficient time for discussion it's now time to formally vote on the proposal.

Please reply to all in this thread with your vote of: YES, NO, or ABSTAIN. A failure to reply will be counted as an abstention. If you vote NO, by our policy you must include the reasons for that vote in your reply. The deadline for votes is *[date and time]*.

If 2/3rds of committers vote YES commit access will be revoked.

The following reasons for revocation have been given in the original proposal or during discussion:

[list of reasons to remove access]

The following reasons for retaining access were discussed:

[list of reasons to retain access]

The counter-argument for each reason for retaining access is:

[list of counter-arguments for retaining access]

8.9.10 Vote Results for Revocation of Commit Access

The voting period for revoking the commit access of *[candidate]* initiated at *[date and time]* is now closed with the following results:

- YES: *[count of yes votes]* (*[% of voters]*)
- NO: *[count of no votes]* (*[% of voters]*)
- ABSTAIN: *[count of abstentions]* (*[% of voters]*)

Based on these results commit access *[is/is NOT]* revoked. The following reasons for retaining commit access were proposed in NO votes:

[list of reasons]

The counter-arguments for each of these reasons are:

[list of counter-arguments]

8.9.11 Notification of Commit Revocation for Detrimental Behavior

After private discussion with you and careful consideration of the situation, the other committers to the OVN project have concluded that it is in the best interest of the project that your commit access to the project repositories be revoked and this has now occurred.

The reasons for this decision are:

[list of reasons for removing access]

While your goals and those of the project no longer appear to be aligned we greatly appreciate all the work you have done for the project and wish you continued success in your future work.

8.10 Authors

The following people authored or signed off on commits in the OVN source code or webpage version control repository. Since OVN originated in the Open vSwitch git repository, this list also includes all of the names in the AUTHORS file at the time OVN was split out from OVS.

Name	Email
Aaron Conole	aconole@redhat.com
Aaron Rosen	arosen@clemson.edu
Aidan Shribman	aidan.shribman@gmail.com
Alan Pevec	alan.pevec@redhat.com
Alexander Duyck	alexander.h.duyck@redhat.com
Alexandru Copot	alex.mihai.c@gmail.com
Alexei Starovoitov	ast@plumgrid.com
Alexey I. Froloff	raorn@raorn.name
Alexey Roytman	roytman@il.ibm.com
Alex Wang	ee07b291@gmail.com
Alfredo Finelli	alf@computationes.de
Alin Balutoiu	abalutoiu@cloudbasesolutions.com
Alin Serdean	aserdean@cloudbasesolutions.com
Ambika Arora	ambika.arora@tcs.com
Amit Bose	bose@noironetworks.com
Amitabha Biswas	azbiswas@gmail.com
Anand Kumar	kumaranand@vmware.com
Andrea Kao	eirinikos@gmail.com
Andreas Karis	akaris@redhat.com
Andrew Evans	
Andrew Beekhof	abeekhof@redhat.com
Andrew Kampjes	a.kampjes@gmail.com
Andrew Lambeth	alambeth@vmware.com
Andre McCurdy	armccurdy@gmail.com
Andy Hill	hillad@gmail.com
Andy Southgate	andy.southgate@citrix.com
Andy Zhou	azhou@ovn.org
Ankur Sharma	ankursharma@vmware.com
Anoob Soman	anoob.soman@citrix.com
Ansis Atteka	aatteka@vmware.com
Antonio Fischetti	antonio.fischetti@intel.com
Antonio Ojea	antonio.ojea.garcia@gmail.com
Anupam Chanda	
Ariel Tubaltsev	atubaltsev@vmware.com
Arnoldo Lutz	arnoldo.lutz.guevara@hpe.com
Artem Teleshev	artem.teleshev@gmail.com
Arun Sharma	arun.sharma@calsoftinc.com
Aryan TaheriMonfared	aryan.taherimonfared@uis.no
Asaf Penso	asafp@mellanox.com
Ashish Varma	ashishvarma.ovs@gmail.com

Continued on next page

Table 2 – continued from previous page

Name	Email
Ashwin Swaminathan	ashwinds@arista.com
Babu Shanmugam	bschanmu@redhat.com
Bala Sankaran	bsankara@redhat.com
Ben Pfaff	blp@ovn.org
Ben Warren	ben@skyportsystems.com
Benli Ye	daniely@vmware.com
Bert Vermeulen	bert@biot.com
Bhanuprakash Bodireddy	bhanuprakash.bodireddy@intel.com
Billy O'Mahony	billy.o.mahony@intel.com
Binbin Xu	xu.binbin1@zte.com.cn
Brian Haley	haleyb.dev@gmail.com
Brian Kruger	bkruger+ovsdev@gmail.com
Bruce Davie	bdavie@vmware.com
Bryan Phillippe	bp@toroki.com
Carlo Andreotti	c.andreotti@m3s.it
Casey Barker	crbarker@google.com
Chandra Sekhar Vejendla	csvejend@us.ibm.com
Christoph Jaeger	cj@linux.com
Chris Wright	chrisw@sous-sol.org
Chuck Short	zulcss@ubuntu.com
Ciara Loftus	ciara.loftus@intel.com
Clint Byrum	clint@fewbar.com
Cong Wang	amwang@redhat.com
Conner Herriges	conner.herriges@ibm.com
Damien Millescamps	damien.millescamps@6wind.com
Damijan Skvarc	damjan.skvarc@gmail.com
Dan Carpenter	dan.carpenter@oracle.com
Dan McGregor	dan.mcgregor@usask.ca
Dan Wendlandt	
Dan Williams	dcbw@redhat.com
Daniel Alvarez	dalvarez@redhat.com
Daniel Borkmann	dborkman@redhat.com
Daniel Hiltgen	daniel@netkine.com
Daniel Roman	
Daniele Di Proietto	daniele.di.proietto@gmail.com
Daniele Venturino	venturino.daniele+ovs@gmail.com
Danny Kukawka	danny.kukawka@bisect.de
Darrell Ball	dlu998@gmail.com
Dave Tucker	dave@dtucker.co.uk
David Erickson	derickso@stanford.edu
David Hill	dhill@redhat.com
David Marchand	david.marchand@redhat.com
David S. Miller	davem@davemloft.net
David Yang	davidy@vmware.com
Dennis Sam	dsam@arista.com
Devendra Naga	devendra.aaru@gmail.com
Dmitry Krivenok	krivenok.dmitry@gmail.com
Dominic Curran	dominic.curran@citrix.com
Dongdong	dongdong1@huawei.com

Continued on next page

Table 2 – continued from previous page

Name	Email
Dongjun	dongj@dtdream.com
Duan Jiong	djduanjiong@gmail.com
Duffie Cooley	
Dustin Lundquist	dustin@null-ptr.net
Ed Maste	emaste@freebsd.org
Ed Swierk	eswierk@skyportsystems.com
Edouard Bourguignon	madko@linuxed.net
Eelco Chaudron	echaudro@redhat.com
Eli Britstein	elibr@mellanox.com
Eric Lapointe	elapointe@corsa.com
Esteban Rodriguez Betancourt	estebarb@hpe.com
Aymerich Edward	edward.aymerich@hpe.com
Edward Tomasz Napierała	trasz@freebsd.org
Eitan Eliahu	eliahue@vmware.com
Eohyung Lee	liquidnuker@gmail.com
Eric Dumazet	edumazet@google.com
Eric Garver	e@erig.me
Eric Sesterhenn	eric.sesterhenn@lsexperts.de
Ethan J. Jackson	ejj@eecs.berkeley.edu
Ethan Rahn	erahn@arista.com
Eziz Durdyyev	ezizdurdy@gmail.com
Fabrizio D'Angelo	fdangelo@redhat.com
Flavio Fernandes	flavio@flaviof.com
Flavio Leitner	fbl@redhat.com
Francesco Fusco	ffusco@redhat.com
François Rigault	frigo@amadeus.com
Frank Wang	wangpeihuixyz@126.com
Frédéric Tobias Christ	fchrist@live.de
Frode Nordahl	frode.nordahl@gmail.com
FUJITA Tomonori	fujita.tomonori@lab.ntt.co.jp
Gabe Beged-Dov	gabe@begeddov.com
Gabriele Cerami	gcerami@redhat.com
Gaetano Catalli	gaetano.catalli@gmail.com
Gal Sagie	gal.sagie@gmail.com
Genevieve LEsperance	glesperance@pivotal.io
Geoffrey Wossum	gwossum@acm.org
Gianluca Merlo	gianluca.merlo@gmail.com
Giuseppe Lettieri	g.lettieri@iet.unipi.it
Glen Gibb	grg@stanford.edu
Gongming Chen	gmingchen@tencent.com
Guoshuai Li	ligs@dtdream.com
Guolin Yang	gyang@vmware.com
Guru Chaitanya Perakam	gperakam@Brocade.com
Gurucharan Shetty	guru@ovn.org
Han Zhou	hzhou@ovn.org
Henry Mai	
Hao Zheng	
Helmut Schaa	helmut.schaa@googlemail.com
Hiteshi Kalra	hiteshi.kalra@tcs.com

Continued on next page

Table 2 – continued from previous page

Name	Email
Huanle Han	hanxueluo@gmail.com
Hui Kang	kangh@us.ibm.com
Hyong Youb Kim	hyonkim@cisco.com
Ian Campbell	Ian.Campbell@citrix.com
Ian Stokes	ian.stokes@intel.com
Ihar Hrachyshka	ihrachys@redhat.com
Ilya Maximets	i.maximets@samsung.com
Iman Tabrizian	tabrizian@outlook.com
Isaku Yamahata	yamahata@valinux.co.jp
Ivan Dyukov	i.dyukov@samsung.com
IWASE Yusuke	iwase.yusuke@gmail.com
Jakub Libosvar	libosvar@redhat.com
Jakub Sitnicki	jsitnicki@gmail.com
James P.	roampune@gmail.com
James Page	james.page@ubuntu.com
James Troup	james.troup@canonical.com
Jamie Lennox	jamielennox@gmail.com
Jan Scheurich	jan.scheurich@ericsson.com
Jan Vansteenkiste	jan@vstone.eu
Jarno Rajahalme	jarno@ovn.org
Jason Kölker	jason@koelker.net
Jason Wessel	jason.wessel@windriver.com
Jasper Capel	jasper@capel.tv
Jean Tourrilhes	jt@hpl.hp.com
Jeremy Stribling	
Jeroen van Bemmel	jvb127@gmail.com
Jesse Gross	jesse@kernel.org
Jian Li	lijian@ooclab.com
Jianbo Liu	jianbol@mellanox.com
Jing Ai	jinga@google.com
Jiri Benc	jbenc@redhat.com
Jochen Friedrich	jochen@scram.de
Joe Perches	joe@perches.com
Joe Stringer	joe@ovn.org
Jonathan Vestin	jonavest@kau.se
Jorge Arturo Sauma Vargas	jorge.sauma@hpe.com
Jun Nakajima	jun.nakajima@intel.com
JunhanYan	juyan@redhat.com
JunoZhu	zhunatuzi@gmail.com
Justin Pettit	jpettit@ovn.org
Kai Li	likailichee@gmail.com
Kaige Fu	fukaige@huawei.com
Keith Amidon	
Ken Ajiro	ajiro@mxw.nes.nec.co.jp
Ken Sanislo	ken@intherack.com
Kenneth Duda	kduda@arista.com
Kentaro Ebisawa	ebiken.g@gmail.com
Keshav Gupta	keshav.gupta@ericsson.com
Kevin Lo	kevlo@FreeBSD.org

Continued on next page

Table 2 – continued from previous page

Name	Email
Kevin Traynor	kevin.traynor@intel.com
Khem Raj	raj.khem@gmail.com
Kmindg G	kmindg@gmail.com
Kris Murphy	kriskend@linux.vnet.ibm.com
Krishna Kondaka	kkondaka@vmware.com
Kyle Mestery	mestery@mestery.com
Kyle Simpson	kyleandrew.simpson@gmail.com
Kyle Upton	kupton@baymicrosystems.com
Lance Richardson	lrichard@redhat.com
Lars Kellogg-Stedman	lars@redhat.com
Lei Huang	huang.f.lei@gmail.com
Leif Madsen	lmadsen@redhat.com
Leo Alterman	
Li RongQing	lirongqing@baidu.com
Lian-min Wang	liang-min.wang@intel.com
Lilijun	jerry.lilijun@huawei.com
Lili Huang	huanglili.huang@huawei.com
Linda Sun	lsun@vmware.com
Lior Neudorfer	lior@guardicore.com
Liu Chang	txfh2007@aliyun.com
Lorand Jakab	lojakab@cisco.com
Lorenzo Bianconi	lorenzo.bianconi@redhat.com
Luca Giraudo	
Lucas Alvares Gomes	lucasagomes@gmail.com
Lucian Petrut	lpetrut@cloudbasesolutions.com
Luigi Rizzo	rizzo@iet.unipi.it
Luis E. P.	l31g@hotmail.com
Lukasz Rzasik	lukasz.rzasik@gmail.com
Madhu Challa	challa@noironetworks.com
Manohar K C	manukc@gmail.com
Manoj Sharma	manoj.sharma@nutanix.com
Marcin Mirecki	mmirecki@redhat.com
Mario Cabrera	mario.cabrera@hpe.com
Mark D. Gray	mark.d.gray@redhat.com
Mark Hamilton	
Mark Kavanagh	mark.b.kavanagh81@gmail.com
Mark Maglana	mmaglana@gmail.com
Mark Michelson	mmichels@redhat.com
Markos Chandras	mchandras@suse.de
Martin Casado	casado@cs.stanford.edu
Martin Fong	mwfong@csl.sri.com
Martino Fornasa	mf@fornasa.it
Martin Xu	martinxu9.ovs@gmail.com
Maryam Tahhan	maryam.tahhan@intel.com
Matteo Croce	mcroce@redhat.com
Matthias May	matthias.may@neratec.com
Mauricio Vásquez	mauricio.vasquezbernal@studenti.polito.it
Maxime Coquelin	maxime.coquelin@redhat.com
Mehak Mahajaon	

Continued on next page

Table 2 – continued from previous page

Name	Email
Mengxin Liu	mengxin@alauda.io
Michael Arnaldi	arnaldimichael@gmail.com
Michal Weglicki	michalx.weglicki@intel.com
Mickey Spiegel	mickeys.dev@gmail.com
Miguel Angel Ajo	majopela@redhat.com
Mijo Safradin	mijo@linux.vnet.ibm.com
Mika Vaisanen	mika.vaisanen@gmail.com
Minoru TAKAHASHI	takahashi.minoru7@gmail.com
Mohammad Heib	mheib@redhat.com
Moshe Levi	moshele@mellanox.com
Murphy McCauley	murphy.mccauley@gmail.com
Natasha Gude	
Neal Shrader	neal@digitalocean.com
Neil McKee	neil.mckee@inmon.com
Neil Zhu	zhuj@centecnetworks.com
Nimay Desai	nimaydesai1@gmail.com
Nithin Raju	nithin@vmware.com
Niti Rohilla	niti.rohilla@tcs.com
Nitin Katiyar	nitin.katiyar@ericsson.com
Numan Siddique	nusiddiq@redhat.com
Ofer Ben-Yacov	ofer.benyacov@gmail.com
Ophir Munk	ophirmu@mellanox.com
Or Gerlitz	ogerlitz@mellanox.com
Ori Shoshan	ori.shoshan@guardicore.com
Padmanabhan Krishnan	kprad1@yahoo.com
Panu Matilainen	pmatilai@redhat.com
Paraneetharan Chandrasekaran	paraneetharanc@gmail.com
Paul Boca	pboca@cloudbasesolutions.com
Paul Fazzone	pfazzone@vmware.com
Paul Ingram	
Paul-Emmanuel Raoul	skyper@skyplabs.net
Pavithra Ramesh	paramesh@vmware.com
Pedro Guimaraes	pedro.guimaraes@canonical.com
Peter Downs	padowns@gmail.com
Philippe Jung	phil.jung@free.fr
Pim van den Berg	pim@nethuis.nl
pritesh	pritesh.kothari@cisco.com
Pravin B Shelar	pshelar@ovn.org
Przemyslaw Szczerbik	przemyslawx.szczerbik@intel.com
Quentin Monnet	quentin.monnet@6wind.com
Qiuyu Xiao	qiuyu.xiao.qyx@gmail.com
Raju Subramanian	
Rami Rosen	ramirose@gmail.com
Ramu Ramamurthy	ramu.ramamurthy@us.ibm.com
Randall Sharo	andall.sharo@navy.mil
Ravi Kerur	Ravi.Kerur@telekom.com
Raymond Burkholder	ray@oneunified.net
Reid Price	
Remko Tronçon	git@el-tramo.be

Continued on next page

Table 2 – continued from previous page

Name	Email
Renat Nurgaliyev	impleman@gmail.com
Rich Lane	rlane@bigswitch.com
Richard Oliver	richard@richard-oliver.co.uk
Rishi Bamba	rishi.bamba@tcs.com
Rob Adams	readams@readams.net
Robert Åkerblom-Andersson	Robert.nr1@gmail.com
Robert Wojciechowicz	robertx.wojciechowicz@intel.com
Rob Hoes	rob.hoes@citrix.com
Robin Brämer	robin.braemer@web.de
Rohith Basavaraja	rohith.basavaraja@gmail.com
Roi Dayan	roid@mellanox.com
Róbert Mulik	robert.mulik@ericsson.com
Romain Lenglet	romain.lenglet@berabera.info
Roni Bar Yanai	roniba@mellanox.com
Russell Bryant	russell@ovn.org
RYAN D. MOATS	rmoats@us.ibm.com
Ryan Goulding	rgouldin@redhat.com
Ryan Wilson	
Sairam Venugopal	vsairam@vmware.com
Sajjad Lateef	
Saloni Jain	saloni.jain@tcs.com
Samuel Ghinet	sghinet@cloudbasesolutions.com
Sanjay Sane	
Saurabh Mohan	saurabh@cplanenetworks.com
Saurabh Shah	
Saurabh Shrivastava	saurabh.shrivastava@nuagenetworks.net
Scott Cheloha	scottcheloha@gmail.com
Scott Lowe	scott.lowe@scottlowe.org
Scott Mann	sdmnix@gmail.com
Selvamuthukumar	smkumar@merunetworks.com
Sha Zhang	zhangsha.zhang@huawei.com
Shad Ansari	shad.ansari@hpe.com
Shan Wei	davidshan@tencent.com
Sharon Krendel	thekafkaf@gmail.com
Shashank Ram	rams@vmware.com
Shashwat Srivastava	shashwat.srivastava@tcs.com
Shih-Hao Li	shihli@vmware.com
Shu Shen	shu.shen@radisys.com
Simon Horman	horms@verge.net.au
Simon Horman	simon.horman@netronome.com
Sorin Vinturis	svinturis@cloudbasesolutions.com
Steffen Gebert	steffen.gebert@informatik.uni-wuerzburg.de
Sten Spans	sten@blinkenlights.nl
Stephane A. Sezer	sas@cd80.net
Stephen Finucane	stephen@that.guru
Steve Ruan	ruansx@cn.ibm.com
Stuart Cardall	developer@it-offshore.co.uk
Sugesh Chandran	sugesh.chandran@intel.com
SUGYO Kazushi	sugyo.org@gmail.com

Continued on next page

Table 2 – continued from previous page

Name	Email
Sven Haardiek	sven.haardiek@uni-muenster.de
Tadaaki Nagao	nagao@stratosphere.co.jp
Terry Wilson	twilson@redhat.com
Tetsuo NAKAGAWA	nakagawa@mxc.nes.nec.co.jp
Thadeu Lima de Souza Cascardo	cascardo@cascardo.eti.br
Thomas F. Herbert	thomasfherbert@gmail.com
Thomas Goirand	zigo@debian.org
Thomas Graf	tgraf@noironetworks.com
Thomas Lacroix	thomas.lacroix@citrix.com
Tim Rozet	trozet@redhat.com
Timo Puha	timox.puha@intel.com
Timothy Redaelli	tredaelli@redhat.com
Todd Deshane	deshantm@gmail.com
Tom Everman	teverman@google.com
Toms Atteka	cpp.code.lv@gmail.com
Torgny Lindberg	torgny.lindberg@ericsson.com
Tsvi Slonim	tsvi@toroki.com
Tuan Nguyen	tuan.nguyen@veriksystems.com
Tyler Coumbes	coumbes@gmail.com
Tony van der Peet	tony.vanderpeet@alliedtelesis.co.nz
Tonghao Zhang	xiangxia.m.yue@gmail.com
Valient Gough	vgough@pobox.com
Vanou Ishii	ishii.vanou@fujitsu.com
Venkata Anil Kommaddi	vkommadi@redhat.com
Venu Iyer	venugopali@nvidia.com
Vishal Deep Ajmera	vishal.deep.ajmera@ericsson.com
Vivien Bernet-Rollande	vbr@soprive.net
Vladislav Odintsov	odivlad@gmail.com
wangqianyu	wang.qianyu@zte.com.cn
Wang Sheng-Hui	shhuiw@gmail.com
Wang Zhike	wangzhike@jd.com
Wei Li	liw@dtdream.com
Wei Yongjun	yjwei@cn.fujitsu.com
Wenyu Zhang	wenyuz@vmware.com
William Fulton	
William Tu	u9012063@gmail.com
Xiao Liang	shaw.leon@gmail.com
xu rong	xu.rong@zte.com.cn
YAMAMOTO Takashi	yamamoto@midokura.com
Yasuhito Takamiya	yasuhito@gmail.com
Yi Li	yili@winhong.com
Yi-Hung Wei	yihung.wei@gmail.com
Yifeng Sun	pkusunyifeng@gmail.com
Yin Lin	linyi@vmware.com
Yu Zhiguo	yuzg@cn.fujitsu.com
Yuanhan Liu	yuanhan.liu@linux.intel.com
Yunjian Wang	wangyunjian@huawei.com
Yousong Zhou	yszhou4tech@gmail.com
Zak Whittington	zwhitt.vmware@gmail.com

Continued on next page

Table 2 – continued from previous page

Name	Email
ZhengLingyun	konghuarukhr@163.com
Zoltán Balogh	zoltan.balogh.eth@gmail.com
Zoltan Kiss	zoltan.kiss@citrix.com
Zongkai LI	zealokii@gmail.com
Zhi Yong Wu	zwu.kernel@gmail.com
Zang MingJie	zealot0630@gmail.com
Zhen Wang	zhewang@nvidia.com
Zhenyu Gao	sysugaozhenyu@gmail.com
ZhiPeng Lu	luzhipeng@uniudc.com
Zhou Yangchao	1028519445@qq.com
aginwala	amginwal@gmail.com
parameswaran krishnamurthy	parkrish@gmail.com
solomon	liwei.solomon@gmail.com
wenxu	wenxu@ucloud.cn
wisd0me	ak47izatool@gmail.com
Xavier Simonart	xsimonar@redhat.com
xieyanker	xjsisnice@gmail.com
xushengping	shengping.xu@huawei.com
yinpeijun	yinpeijun@huawei.com
zangchuanqiang	zangchuanqiang@huawei.com
zhang yanxian	zhangyanxian@pmlabs.com.cn
zhaojingjing	zhao.jingjing1@zte.com.cn
zhongbaisong	zhongbaisong@huawei.com
zhaozhanxu	zhaozhanxu@163.com

The following additional people are mentioned in commit logs as having provided helpful bug reports or suggestions.

Name	Email
Aaron M. Ucko	ucko@debian.org
Abhinav Singhal	Abhinav.Singhal@spirent.com
Adam Heath	doogie@brainfood.com
Ahmed Bilal	numan252@gmail.com
Alan Kayahan	hsykay@gmail.com
Alan Shieh	
Alban Browaeys	prahal@yahoo.com
Alex Yip	
Alexey I. Froloff	raorn@altlinux.org
Amar Padmanabhan	
Amey Bhide	
Amre Shakimov	ashakimov@vmware.com
André Ruß	andre.russ@hybris.com
Andreas Beckmann	debian@abeckmann.de
Andrei Andone	andrei.andone@softvision.ro
Andrey Korolyov	andrey@xdel.ru
Anil Jangam	anilj.mailing@gmail.com
Anshuman Manral	anshuman.manral@outlook.com
Anton Matsiuk	anton.matsiuk@gmail.com
Anup Khadka	khadka.py@gmail.com
Anuprem Chalvadi	achalvadi@vmware.com

Continued on next page

Table 3 – continued from previous page

Name	Email
Ariel Tubaltsev	atubaltsev@vmware.com
Arkajit Ghosh	arkajit.ghosh@tcs.com
Atzm Watanabe	atzm@stratosphere.co.jp
Aurélien Poulain	aurepoulain@viacesi.fr
Bastian Blank	waldi@debian.org
Ben Basler	
Bhargava Shastry	bshastry@sec.t-labs.tu-berlin.de
Bob Ball	bob.ball@citrix.com
Brad Hall	
Brad Cowie	brad@wand.net.nz
Brailey Josh	josh@faucet.nz
Brandon Heller	brandonh@stanford.edu
Brendan Kelley	
Brent Salisbury	brent.salisbury@gmail.com
Brian Field	Brian_Field@cable.comcast.com
Bryan Fulton	
Bryan Osoro	
Cedric Hobbs	
Chris Hydon	chydon@aristanetworks.com
Christian Stigen Larsen	cslarsen@gmail.com
Christopher Paggen	cpaggen@cisco.com
Chunhe Li	lichunhe@huawei.com
Daniel Badea	daniel.badea@windriver.com
Darragh O'Reilly	darragh.oreilly@hpe.com
Dave Walker	DaveWalker@ubuntu.com
David Evans	davidjoshuaevans@gmail.com
David Palma	palma@onesource.pt
David van Moolenbroek	dvmoolenbroek@aimvalley.nl
Derek Cormier	derek.cormier@lab.ntt.co.jp
Dhaval Badiani	dbadiani@vmware.com
DK Moon	
Ding Zhi	zhi.ding@6wind.com
Dong Jun	dongj@dtdream.com
Dustin Spinhirne	dspinhirne@vmware.com
Edwin Chiu	echiu@vmware.com
Eivind Bulie Haanaes	
Enas Ahmad	enas.ahmad@kaust.edu.sa
Eric Lopez	
Frido Roose	fr.roose@gmail.com
Gaetano Catalli	gaetano.catalli@gmail.com
Gavin Remaley	gavin_remaley@selinc.com
Georg Schmuecking	georg.schmuecking@ericsson.com
George Shuklin	amarao@desunote.ru
Gerald Rogers	gerald.rogers@intel.com
Ghanem Bahri	bahri.ghanem@gmail.com
Giuseppe de Candia	giuseppe.decandia@gmail.com
Gordon Good	ggood@vmware.com
Greg Dahlman	gdahlman@hotmail.com
Greg Rose	gvrose8192@gmail.com

Continued on next page

Table 3 – continued from previous page

Name	Email
Gregor Schaffrath	grsch@net.t-labs.tu-berlin.de
Gregory Smith	gasmith@nutanix.com
Guolin Yang	gyang@vmware.com
Gur Stavi	gstavi@mrv.com
Harish Kanakaraju	hkanakaraju@vmware.com
Hari Sasank Bhamidipalli	hbhamidi@cisco.com
Hassan Khan	hassan.khan@seecs.edu.pk
Hector Oron	hector.oron@gmail.com
Hemanth Kumar Mantri	mantri@nutanix.com
Henrik Amren	
Hiroshi Tanaka	
Hiroshi Miyata	miyahiro.dazu@gmail.com
Hsin-Yi Shen	shenh@vmware.com
Hui Xiang	xianghuir@gmail.com
Hyojoon Kim	joonk@gatech.edu
Igor Ganichev	
Igor Sever	igor@xorops.com
Jacob Cherkas	cherkasj@vmware.com
Jad Naous	jnaous@gmail.com
Jamal Hadi Salim	hadi@cyberus.ca
James Schmidt	jschmidt@vmware.com
Jan Medved	jmedved@juniper.net
Janis Hamme	janis.hamme@student.kit.edu
Jari Sundell	sundell.software@gmail.com
Javier Albornoz	javier.albornoz@hpe.com
Jed Daniels	openvswitch@jeddaniels.com
Jeff Merrick	jmerrick@vmware.com
Jeongkeun Lee	jklee@hp.com
Jian Qiu	swordqiu@gmail.com
Joan Cirer	joan@ev0.net
John Darrington	john@darrington.wattle.id.au
John Galgay	john@galgay.net
John Hurley	john.hurley@netronome.com
John Reumann	nofutznetworks@gmail.com
Karthik Sundaravel	ksundara@redhat.com
Kashyap Thimmaraju	kashyap.thimmaraju@sec.t-labs.tu-berlin.de
Keith Holleman	hollemanietf@gmail.com
Kevin Lin	kevinlin@berkeley.edu
K	k940545@hotmail.com
Kevin Mancuso	kevin.mancuso@rackspace.com
Kiran Shanbhog	kiran@vmware.com
Kirill Kabardin	
Kirkland Spector	kspector@salesforce.com
Koichi Yagishita	yagishita.koichi@jrc.co.jp
Konstantin Khorenko	khorenko@openvz.org
Kris zhang	zhang.kris@gmail.com
Krishna Miriyala	miriyalak@vmware.com
Krishna Mohan Elluru	elluru.kri.mohan@hpe.com
László Sürü	laszlo.suru@ericsson.com

Continued on next page

Table 3 – continued from previous page

Name	Email
Len Gao	leng@vmware.com
Logan Rosen	logatronico@gmail.com
Luca Falavigna	dktrkranz@debian.org
Luiz Henrique Ozaki	luiz.ozaki@gmail.com
Manpreet Singh	er.manpreet25@gmail.com
Marco d'Itri	md@Linux.IT
Martin Vizvary	vizvary@ics.muni.cz
Marvin Pascual	marvin@pascual.com.ph
Maxime Brun	m.brun@alphalink.fr
Madhu Venugopal	mavenugo@gmail.com
Michael A. Collins	mike.a.collins@ark-net.org
Michael Ben-Ami	mbenami@digitalocean.com
Michael Hu	humichael@vmware.com
Michael J. Smalley	michaeljsmalley@gmail.com
Michael Mao	
Michael Shigorin	mike@osdn.org.ua
Michael Stapelberg	stapelberg@debian.org
Mihir Gangar	gangarm@vmware.com
Mike Bursell	mike.bursell@citrix.com
Mike Kruze	
Mike Qing	mqing@vmware.com
Min Chen	ustcer.tonychan@gmail.com
Mikael Doverhag	
Mircea Ulinic	ping@mirceaulinic.net
Mrinmoy Das	mrdas@ixiacom.com
Muhammad Shahbaz	mshahbaz@cs.princeton.edu
Murali R	muralirdev@gmail.com
Nagi Reddy Jonnala	njonnala@Brocade.com
Niels van Adrichem	N.L.M.vanAdrichem@tudelft.nl
Niklas Andersson	
Oscar Wilde	xdxiaobin@gmail.com
Pankaj Thakkar	pthakkar@vmware.com
Pasi Kärkkäinen	pasik@iki.fi
Patrik Andersson R	patrik.r.andersson@ericsson.com
Paul Greenberg	
Paulo Cravero	pcravero@as2594.net
Pawan Shukla	shuklap@vmware.com
Periyasamy Palanisamy	periyasamy.palanisamy@ericsson.com
Peter Amidon	peter@picnicpark.org
Peter Balland	
Peter Phaal	peter.phaal@inmon.com
Prabina Pattnaik	Prabina.Pattnaik@nechclst.in
Pratap Reddy	
Ralf Heiringhoff	ralf@frosty-geek.net
Ram Jothikumar	
Ramana Reddy	gtvrreddy@gmail.com
Ray Li	rayli1107@gmail.com
Richard Theis	rtheis@us.ibm.com
RishiRaj Maulick	rishi.raj2509@gmail.com

Continued on next page

Table 3 – continued from previous page

Name	Email
Rob Sherwood	rob.sherwood@bigswitch.com
Robert Strickler	anomalyst@gmail.com
Roger Leigh	rleigh@codelibre.net
Rogério Vinhal Nunes	
Roman Sokolkov	rsokolkov@gmail.com
Ronaldo A. Ferreira	ronaldof@CS.Princeton.EDU
Ronny L. Bull	bullrl@clarkson.edu
Sandeep Kumar	sandeep.kumar16@tcs.com
Sander Eikelenboom	linux@eikelenboom.it
Saul St. John	sstjohn@cs.wisc.edu
Scott Hendricks	
Sean Brady	sbrady@gtfservices.com
Sebastian Andrzej Siewior	sebastian@breakpoint.cc
Sébastien RICCIO	sr@swisscenter.com
Shweta Seth	shwseth@cisco.com
Simon Jouet	simon.jouet@gmail.com
Spiro Kourtessis	spiro@vmware.com
Sridhar Samudrala	samudrala.sridhar@gmail.com
Srini Seetharaman	seethara@stanford.edu
Sabyasachi Sengupta	Sabyasachi.Sengupta@alcatel-lucent.com
Salvatore Cambria	salvatore.cambria@citrix.com
Soner Sevinc	sevincs@vmware.com
Stepan Andrushko	stepanx.andrushko@intel.com
Stephen Hemminger	shemminger@vyatta.com
Stuart Cardall	developer@it-offshore.co.uk
Suganya Ramachandran	suganyar@vmware.com
Sundar Nadathur	undar.nadathur@intel.com
Taekho Nam	thnam@smartx.kr
Takayuki HAMA	t-hama@cb.jp.nec.com
Teemu Koponen	
Thomas Morin	thomas.morin@orange.com
Timothy Chen	
Torbjorn Tornkvist	kruskakli@gmail.com
Tulio Ribeiro	tribeiro@lasige.di.fc.ul.pt
Tytus Kurek	Tytus.Kurek@pega.com
Valentin Bud	valentin@hackaserver.com
Vasiliy Tolstov	v.tolstov@selfip.ru
Vasu Dasari	vdasari@gmail.com
Vinllen Chen	cvinllen@gmail.com
Vishal Swarankar	vishal.swarnkar@gmail.com
Vjekoslav Brajkovic	balkan@cs.washington.edu
Voravit T.	voravit@kth.se
Yeming Zhao	zhaoyeming@gmail.com
Yi Ba	yby.developer@yahoo.com
Ying Chen	yingchen@vmware.com
Yongqiang Liu	liuyq7809@gmail.com
ZHANG Zhiming	zhangzhiming@yunshan.net.cn
Zhangguanghui	zhang.guanghui@h3c.com
Ziyou Wang	ziyouw@vmware.com

Continued on next page

Table 3 – continued from previous page

Name	Email
ankur dwivedi	ankurengg2003@gmail.com
chen zhang	3zhangchen9211@gmail.com
james hopper	jameshopper@email.com
kk yap	yapkke@stanford.edu
likunyun	kunyunli@hotmail.com
meishengxin	meishengxin@huawei.com
neeraj mehta	mehtaneeraj07@gmail.com
rahim entezari	rahim.entezari@gmail.com
shaoke xi	xishaoke.xsk@gmail.com
shivani dommeti	shivani.dommeti@gmail.com
weizj	34965317@qq.com
	zhaojun12@outlook.com
(Crab)	fqs888@126.com
	fortitude.zhang@gmail.com
	hujingfei914@msn.com
	zhangwqh@126.com
	zhangqiang@meizu.com

Thanks to all Open vSwitch and OVN contributors. If you are not listed above but believe that you should be, please write to dev@openvswitch.org.

8.11 Committers

OVN committers are the people who have been granted access to push changes to to the OVN git repository.

The responsibilities of an OVN committer are documented here.

The process for adding or removing committers is documented here.

This is the current list of active OVN committers:

Table 4: OVS Maintainers

Name	Email
Gurucharan Shetty	guru@ovn.org
Han Zhou	hzhou@ovn.org
Justin Pettit	jpettit@ovn.org
Leonid Ryzhyk	lryzhyk@vmware.com
Mark Michelson	mmichels@redhat.com
Numan Siddique	nusddiq@redhat.com
Russell Bryant	russell@ovn.org

The project also maintains a list of Emeritus Committers (or Maintainers). More information about Emeritus Committers can be found here.

Table 5: OVS Emeritus Maintainers

Name	Email
Ben Pfaff	blp@ovn.org

8.12 How OVN's Documentation Works

This document provides a brief overview on how the documentation build system within OVN works. This is intended to maximize the "bus factor" and share best practices with other projects.

8.12.1 reStructuredText and Sphinx

Nearly all of OVN's documentation is written in reStructuredText, with man pages being the sole exception. Of this documentation, most of it is fed into Sphinx, which provides not only the ability to convert rST to a variety of other output formats but also allows for things like cross-referencing and indexing. for more information on the two, refer to the *OVN Documentation Style*.

8.12.2 ovs-sphinx-theme

The documentation uses its own theme, *ovs-sphinx-theme*, which can be found on GitHub and is published on pypi. This is shared by Open vSwitch and OVN. It is packaged separately to ensure all documentation gets the latest version of the theme (assuming there are no major version bumps in that package). If building locally and the package is installed, it will be used. If the package is not installed, Sphinx will fallback to the default theme.

The package is currently maintained by Stephen Finucane and Russell Bryant.

8.12.3 Read the Docs

The documentation is hosted on readthedocs.org and a CNAME redirect is in place to allow access from docs.ovn.org. *Read the Docs* provides a couple of nifty features for us, such as automatic building of docs whenever there are changes and versioning of documentation.

The *Read the Docs* project is currently maintained by Stephen Finucane, Russell Bryant and Ben Pfaff.

8.12.4 ovn.org

The sources for ovn.org are maintained separately from docs.ovn.org. For modifications to this site, refer to the GitHub project.

8.13 OVS Submodule

Prior to 2020, OVN did not exist as its own repo. Instead, OVN was a subdirectory within OVS. OVN grew up being closely intertwined with OVS. Compiling OVS would also compile OVN. OVN used OVS libraries directly, and there was no concern about trying to maintain any level of compatibility between OVS and OVN since they were the same codebase.

In 2020, OVN was split off from OVS. This meant that it became necessary to consider compatibility between OVS and OVN. At compile time, we use a submodule to ensure that OVS libraries that OVN relies on will behave as expected. Runtime compatibility is a separate topic outside the scope of this document.

8.13.1 Developing with the OVS submodule

Most OVN development will happen independently of the OVS submodule. However, there may be times that in order to make a change in OVN, an accompanying change is required in OVS as well. For instance, it may be that a change to OVSDB's client API is required for OVN to fix a bug.

In this situation, make the necessary OVS change first and submit this fix to OVS based on their current code submission guidelines. Once the change has been accepted by OVS, then you can submit an OVN patch that includes changing the submodule to point at the OVS commit where your change was accepted.

8.13.2 Submodules for releases

For OVN releases, it is preferred for the OVS submodule to point to a stable release branch of OVS. Therefore, as part of the release process for OVN, we will point the submodule to the latest stable branch of OVS before releasing.

The exception to this is if the current OVS submodule is pointing to a commit that is not in a current stable branch of OVS. In that case, the submodule will continue to point to that particular commit. We may, however, bump the submodule to the next stable branch of OVS at a later time.

As an example, let's assume that the OVS commit history looks something like this in the main branch:

```
(Newest)
Commit 3
   |
   |
Commit 2 (used to create OVS branch-Y)
   |
   |
Commit 1
(Oldest)
```

Let's say that we are planning to release OVN version X. At this point, the submodule is pointing to Commit 1. As part of the release process, we will bump the OVS submodule in OVN to point to Commit 2, or more likely the tip of OVS branch-Y. This way, the released version of OVN is based on a stable release branch of OVS, and it has all of the necessary changes that we require.

What if the OVS submodule currently points to Commit 3, though? There is no stable branch that exists after this commit. In this case, we have two choices:

 # Create OVN release X and point the OVS submodule to Commit 3. At a later time, if it makes sense to do so, we may bump the submodule to OVS branch-Z when it is released, since Commit 3 will be included in that branch.

 # If Commit 3 is a bug fix in OVS, then we can try to ensure that Commit 3 gets backported to OVS branch-Y, and then point the submodule commit to the tip of OVS branch-Y.

For choice 1, the decision of whether to update the submodule commit to OVS branch-Z is based on several factors.

- Is OVN release X still being supported?
- Is there any known benefit to updating the submodule? E.g., are there performance improvements we could take advantage of by updating the submodule?
- Is there risk in updating the submodule?

For an LTS of OVN, we might update the submodule several times during its lifetime as more new OVS branches are released. For a standard release, it is less likely that we will update the OVS submodule during the standard release's lifetime.