## NAME

ovn-northd − Open Virtual Network central control daemon

## SYNOPSIS

**ovn−northd** [*options*]

## DESCRIPTION

**ovn−northd** is a centralized daemon responsible for translating the high-level OVN configuration into logical configuration consumable by daemons such as **ovn−controller**. It translates the logical network configuration in terms of conventional network concepts, taken from the OVN Northbound Database (see **ovn−nb**(5)), into logical datapath flows in the OVN Southbound Database (see **ovn−sb**(5)) below it.

## CONFIGURATION

**ovn−northd** requires a connection to the Northbound and Southbound databases. The default is **db.sock** in the local Open vSwitch's "run" directory. This may be overridden with the following commands:

- **−−ovnnb−db=***database*

  The database containing the OVN Northbound Database.

- **−−ovsnb−db=***database*

  The database containing the OVN Southbound Database.

The *database* argument must take one of the following forms:

- **ssl:***ip***:***port*

  The specified SSL *port* on the host at the given *ip*, which must be expressed as an IP address (not a DNS name) in IPv4 or IPv6 address format. If *ip* is an IPv6 address, then wrap *ip* with square brackets, e.g.: **ssl:[::1]:6640**. The **−−private−key**, **−−certificate**, and **−−ca−cert** options are mandatory when this form is used.

- **tcp:***ip***:***port*

  Connect to the given TCP *port* on *ip*, where *ip* can be IPv4 or IPv6 address. If *ip* is an IPv6 address, then wrap *ip* with square brackets, e.g.: **tcp:[::1]:6640**.

- **unix:***file*

  On POSIX, connect to the Unix domain server socket named *file*.

  On Windows, connect to a localhost TCP port whose value is written in *file*.

## RUNTIME MANAGEMENT COMMANDS

**ovs−appctl** can send commands to a running **ovn−northd** process. The currently supported commands are described below.

**exit** Causes **ovn−northd** to gracefully terminate.

## LOGICAL FLOW TABLE STRUCTURE

One of the main purposes of **ovn−northd** is to populate the **Logical_Flow** table in the **OVN_Southbound** database. This section describes how **ovn−northd** does this for logical datapaths.

### Ingress Table 0: Admission Control and Ingress Port Security

Ingress table 0 contains these logical flows:

- Priority 100 flows to drop packets with VLAN tags or multicast Ethernet source addresses.

- Priority 50 flows that implement ingress port security for each enabled logical port. For logical ports on which port security is enabled, these match the **inport** and the valid **eth.src** address(es) and advance only those packets to the next flow table. For logical ports on which port security is not enabled, these advance all packets that match the **inport**.

There are no flows for disabled logical ports because the default-drop behavior of logical flow tables causes packets that ingress from them to be dropped.

**Ingress table 1: from−lport** ACLs

Logical flows in this table closely reproduce those in the **ACL** table in the **OVN_Northbound** database for the **from−lport** direction. **allow** and **allow−related** ACLs translate into logical flows with the **next;** action, others to **drop;**. The **priority** values from the **ACL** table are used directly.

Ingress table 1 also contains a priority 0 flow with action **next;**, so that ACLs allow packets by default.

**Ingress Table 2: Destination Lookup**

This table implements switching behavior. It contains these logical flows:

- A priority−100 flow that outputs all packets with an Ethernet broadcast or multicast **eth.dst** to the **MC_FLOOD** multicast group, which **ovn−northd** populates with all enabled logical ports.
- One priority−50 flow that matches each known Ethernet address against **eth.dst** and outputs the packet to the single associated output port.
- One priority−0 fallback flow that matches all packets and outputs them to the **MC_UNKNOWN** multicast group, which **ovn−northd** populates with all enabled logical ports that accept unknown destination packets. As a small optimization, if no logical ports accept unknown destination packets, **ovn−northd** omits this multicast group and logical flow.

**Egress Table 0: to−lport** ACLs

This is similar to ingress table 1 except for **to−lport** ACLs.

**Egress Table 1: Egress Port Security**

This is similar to the ingress port security logic in ingress table 0, but with important differences. Most obviously, **outport** and **eth.dst** are checked instead of **inport** and **eth.src**. Second, packets directed to broadcast or multicast **eth.dst** are always accepted instead of being subject to the port security rules; this is implemented through a priority−100 flow that matches on **eth.dst[40]** with action **output;**. Finally, to ensure that even broadcast and multicast packets are not delivered to disabled logical ports, a priority−150 flow for each disabled logical **outport** overrides the priority−100 flow with a **drop;** action.