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 switch and router logical datapaths.

Logical Switch 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.mcast 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.

**Logical Router Datapaths**

**Ingress Table 0: L2 Admission Control**

This table drops packets that the router shouldn’t see at all based on their Ethernet headers. It contains the following flows:

- Priority–100 flows to drop packets with VLAN tags or multicast Ethernet source addresses.
- For each enabled router port P with Ethernet address E, a priority–50 flow that matches inport == P && (eth.mcast || eth.dst == E), with action next;

Other packets are implicitly dropped.

**Ingress Table 1: IP Input**

This table is the core of the logical router datapath functionality. It contains the following flows to implement very basic IP host functionality.

- L3 admission control: A priority–220 flow drops packets that match any of the following:
  - ip4.src[28..31] == 0xe (multicast source)
  - ip4.src == 255.255.255.255 (broadcast source)
  - ip4.src == 127.0.0.0/8 || ip4.dst == 127.0.0.0/8 (localhost source or destination)
• \( \text{ip4.src} == 0.0.0.0/8 \| \text{ip4.dst} == 0.0.0.0/8 \) (zero network source or destination)
• \( \text{ip4.src} \) is any IP address owned by the router.
• \( \text{ip4.src} \) is the broadcast address of any IP network known to the router.

- ICMP echo reply. These flows reply to ICMP echo requests received for the router’s IP address. Let \( A \) be an IP address or broadcast address owned by a router port. Then, for each \( A \), a priority−210 flow matches on \( \text{ip4.dst} == A \) and \( \text{icmp4.type} == 8 \&\& \text{icmp4.code} == 0 \) (ICMP echo request). These flows use the following actions where, if \( A \) is unicast, then \( S \) is \( A \), and if \( A \) is broadcast, \( S \) is the router’s IP address in \( A \)’s network:

\[
\begin{align*}
\text{ip4.dst} & = \text{ip4.src}; \\
\text{ip4.src} & = S; \\
\text{ip4.ttl} & = 255; \\
\text{icmp4.type} & = 0; \\
\text{next};
\end{align*}
\]

Similar flows match on \( \text{ip4.dst} == 255.255.255.255 \) and each individual \( \text{inport} \), and use the same actions in which \( S \) is a function of \( \text{inport} \).

- ARP reply. These flows reply to ARP requests for the router’s own IP address. For each router port \( P \) that owns IP address \( A \) and Ethernet address \( E \), a priority−210 flow matches \( \text{inport} == P \&\& \text{arp.tpa} == A \&\& \text{arp.op} == 1 \) (ARP request) with the following actions:

\[
\begin{align*}
\text{eth.dst} & = \text{eth.src}; \\
\text{eth.src} & = E; \\
\text{arp.op} & = 2; /* ARP reply. */ \\
\text{arp.tha} & = \text{arp.sha}; \\
\text{arp.sha} & = E; \\
\text{arp.tpa} & = \text{arp.spa}; \\
\text{arp.spa} & = A; \\
\text{outport} & = P; \\
\text{inport} & = 0; /* Allow sending out inport. */ \\
\text{output};
\end{align*}
\]

- UDP port unreachable. These flows generate ICMP port unreachable messages in reply to UDP datagrams directed to the router’s IP address. The logical router doesn’t accept any UDP traffic so it always generates such a reply. These flows should not match IP fragments with nonzero offset.

Details TBD.

- TCP reset. These flows generate TCP reset messages in reply to TCP datagrams directed to the router’s IP address. The logical router doesn’t accept any TCP traffic so it always generates such a reply.

These flows should not match IP fragments with nonzero offset.

Details TBD.

- Protocol unreachable. These flows generate ICMP protocol unreachable messages in reply to packets directed to the router’s IP address on IP protocols other than UDP, TCP, and ICMP.

These flows should not match IP fragments with nonzero offset.

Details TBD.

- Drop other IP traffic to this router. These flows drop any other traffic destined to an IP address of this router that is not already handled by one of the flows above. For each IP address \( A \) owned by the router, a priority−200 flow matches \( \text{ip4.dst} == A \) and drops the traffic.
The flows above handle all of the traffic that might be directed to the router itself. The following flows (with lower priorities) handle the remaining traffic, potentially for forwarding:

- Drop Ethernet local broadcast. A priority−190 flow with match `eth.bcast` drops traffic destined to the local Ethernet broadcast address. By definition this traffic should not be forwarded.
- ICMP time exceeded. For each router port \( P \), whose IP address is \( A \), a priority−180 flow with match `in_port == P && ip4.ttl == \{0, 1\} && !ip.later_frag` matches packets whose TTL has expired, with the following actions to send an ICMP time exceeded reply:

\[
\text{icmp4} \\
\text{    icmp4.type} = 11; /* Time exceeded. */ \\
\text{    icmp4.code} = 0; /* TTL exceeded in transit. */ \\
\text{    ip4.dst} = \text{ip4.src}; \\
\text{    ip4.src} = A; \\
\text{    ip4.ttl} = 255; \\
\text{    next}; \\
\]

- TTL discard. A priority−170 flow with match `ip4.ttl < 2` and actions `drop`; drops other packets whose TTL has expired, that should not receive a ICMP error reply.

**Ingress Table 2: IP Routing**

A packet that arrives at this table is an IP packet that should be routed to the address in `ip4.dst`. This table implements IP routing, setting `reg0` to the next-hop IP address (leaving `ip4.dst`, the packet’s final destination, unchanged) and advances to the next table for ARP resolution.

This table contains the following logical flows:

- Routing table. For each route to IPv4 network \( N \) with netmask \( M \), a logical flow with match `ip4.dst == N/M`, whose priority is the number of 1-bits in \( M \), has the following actions:

\[
\text{ip4.ttl}--; \\
\text{reg0} = G; \\
\text{next}; \\
\]

(Ingress table 1 already verified that `ip4.ttl--;` will not yield a TTL exceeded error.)

If the route has a gateway, \( G \) is the gateway IP address, otherwise it is `ip4.dst`.

- Destination unreachable. For each router port \( P \), which owns IP address \( A \), a priority−0 logical flow with match `in_port == P && !ip.later_frag && !icmp` has the following actions:

\[
\text{icmp4} \\
\text{    icmp4.type} = 3; /* Destination unreachable. */ \\
\text{    icmp4.code} = 0; /* Network unreachable. */ \\
\text{    ip4.dst} = \text{ip4.src}; \\
\text{    ip4.src} = A; \\
\text{    ip4.ttl} = 255; \\
\text{    next(2)}; \\
\]

(The `!icmp` check prevents recursion if the destination unreachable message itself cannot be routed.)

These flows are omitted if the logical router has a default route, that is, a route with netmask 0.0.0.0.
Ingress Table 3: ARP Resolution

Any packet that reaches this table is an IP packet whose next-hop IP address is in reg0. (ip4.dst is the final destination.) This table resolves the IP address in reg0 into an output port in outport and an Ethernet address in eth.dst, using the following flows:

- Known MAC bindings. For each IP address A whose host is known to have Ethernet address HE and reside on router port P with Ethernet address PE, a priority−200 flow with match reg0 == A has the following actions:

  ```
  eth.src = PE;
  eth.dst = HE;
  outport = P;
  output;
  ```

  MAC bindings can be known statically based on data in the OVN_Northbound database. For router ports connected to logical switches, MAC bindings can be known statically from the addresses column in the Logical_Port table. For router ports connected to other logical routers, MAC bindings can be known statically from the mac and network column in the Logical_Router_Port table.

- Unknown MAC bindings. For each non-gateway route to IPv4 network N with netmask M on router port P that owns IP address A and Ethernet address E, a logical flow with match ip4.dst == N/M, whose priority is the number of 1-bits in M, has the following actions:

  ```
  arp {
    eth.dst = ff:ff:ff:ff:ff:ff;
    eth.src = E;
    arp.sha = E;
    arp.tha = 00:00:00:00:00:00;
    arp.spa = A;
    arp.tpa = ip4.dst;
    arp.op = 1; /* ARP request. */
    outport = P;
    output;
  }
  ```

  TBD: How to install MAC bindings when an ARP response comes back. (Implement a "learn" action?)

Egress Table 0: Delivery

Packets that reach this table are ready for delivery. It contains priority−100 logical flows that match packets on each enabled logical router port, with action output;.