

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 defaults are **ovnnb_db.sock** and **ovnsb_db.sock** respectively 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.
- **--ovnsb-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 - L2

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: Ingress Port Security - IP

Ingress table 1 contains these logical flows:

- For each element in the port security set having one or more IPv4 or IPv6 addresses (or both),
 - Priority 90 flow to allow IPv4 traffic if it has IPv4 addresses which match the **inport**, valid **eth.src** and valid **ip4.src** address(es).
 - Priority 90 flow to allow IPv4 DHCP discovery traffic if it has a valid **eth.src**. This is necessary since DHCP discovery messages are sent from the unspecified IPv4 address (0.0.0.0) since the IPv4 address has not yet been assigned.
 - Priority 90 flow to allow IPv6 traffic if it has IPv6 addresses which match the **inport**, valid **eth.src** and valid **ip6.src** address(es).
 - Priority 90 flow to allow IPv6 DAD (Duplicate Address Detection) traffic if it has a valid **eth.src**. This is necessary since DAD include requires joining an multicast group and sending neighbor solicitations for the newly assigned address. Since no address is yet assigned, these are sent from the unspecified IPv6 address (::).
 - Priority 80 flow to drop IP (both IPv4 and IPv6) traffic which match the **inport** and valid **eth.src**.
- One priority-0 fallback flow that matches all packets and advances to the next table.

Ingress Table 2: Ingress Port Security - Neighbor discovery

Ingress table 2 contains these logical flows:

- For each element in the port security set,
 - Priority 90 flow to allow ARP traffic which match the **inport** and valid **eth.src** and **arp.sha**. If the element has one or more IPv4 addresses, then it also matches the valid **arp.spa**.
 - Priority 90 flow to allow IPv6 Neighbor Solicitation and Advertisement traffic which match the **inport**, valid **eth.src** and **nd.sll/nd.tll**. If the element has one or more IPv6 addresses, then it also matches the valid **nd.target** address(es) for Neighbor Advertisement traffic.
 - Priority 80 flow to drop ARP and IPv6 Neighbor Solicitation and Advertisement traffic which match the **inport** and valid **eth.src**.
- One priority-0 fallback flow that matches all packets and advances to the next table.

Ingress Table 3: from-lport Pre-ACLs

This table prepares flows for possible stateful ACL processing in ingress table **ACLs**. It contains a priority-0 flow that simply moves traffic to the next table. If stateful ACLs are used in the logical datapath, a priority-100 flow is added that sets a hint (with **reg0[0] = 1; next;**) for table **Pre-stateful** to send IP packets to the connection tracker before eventually advancing to ingress table **ACLs**.

Ingress Table 4: Pre-LB

This table prepares flows for possible stateful load balancing processing in ingress table **LB** and **Stateful**. It contains a priority-0 flow that simply moves traffic to the next table. If load balancing rules with virtual IP addresses (and ports) are configured in **OVN_Northbound** database for a logical datapath, a priority-100 flow is added for each configured virtual IP address **VIP** with a match **ip && ip4.dst == VIP** that sets an action **reg0[0] = 1; next;** to act as a hint for table **Pre-stateful** to send IP packets to the connection tracker for packet de-fragmentation before eventually advancing to ingress table **LB**.

Ingress Table 5: Pre-stateful

This table prepares flows for all possible stateful processing in next tables. It contains a priority-0 flow that simply moves traffic to the next table. A priority-100 flow sends the packets to connection tracker based on a hint provided by the previous tables (with a match for **reg0[0] == 1**) by using the **ct_next**; action.

Ingress table 6: 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. The **priority** values from the **ACL** table have a limited range and have 1000 added to them to leave room for OVN default flows at both higher and lower priorities.

- **allow** ACLs translate into logical flows with the **next**; action. If there are any stateful ACLs on this datapath, then **allow** ACLs translate to **ct_commit; next**; (which acts as a hint for the next tables to commit the connection to conntrack),
- **allow-related** ACLs translate into logical flows with the **ct_commit(ct_label=0/1); next**; actions for new connections and **reg0[1] = 1; next**; for existing connections.
- Other ACLs translate to **drop**; for new or untracked connections and **ct_commit(ct_label=1/1);** for known connections. Setting **ct_label** marks a connection as one that was previously allowed, but should no longer be allowed due to a policy change.

This table also contains a priority 0 flow with action **next**;, so that ACLs allow packets by default. If the logical datapath has a stateful ACL, the following flows will also be added:

- A priority-1 flow that sets the hint to commit IP traffic to the connection tracker (with action **reg0[1] = 1; next**;). This is needed for the default allow policy because, while the initiator's direction may not have any stateful rules, the server's may and then its return traffic would not be known and marked as invalid.
- A priority-65535 flow that allows any traffic in the reply direction for a connection that has been committed to the connection tracker (i.e., established flows), as long as the committed flow does not have **ct_label[0]** set. We only handle traffic in the reply direction here because we want all packets going in the request direction to still go through the flows that implement the currently defined policy based on ACLs. If a connection is no longer allowed by policy, **ct_label[0]** will get set and packets in the reply direction will no longer be allowed, either.
- A priority-65535 flow that allows any traffic that is considered related to a committed flow in the connection tracker (e.g., an ICMP Port Unreachable from a non-listening UDP port), as long as the committed flow does not have **ct_label[0]** set.
- A priority-65535 flow that drops all traffic marked by the connection tracker as invalid.
- A priority-65535 flow that drops all traffic in the reply direction with **ct_label[0]** set meaning that the connection should no longer be allowed due to a policy change. Packets in the request direction are skipped here to let a newly created ACL re-allow this connection.

Ingress Table 7: LB

It contains a priority-0 flow that simply moves traffic to the next table. For established connections a priority 100 flow matches on **ct.est && !ct.rel && !ct.new && !ct.inv** and sets an action **reg0[2] = 1; next**; to act as a hint for table **Stateful** to send packets through connection tracker to NAT the packets. (The packet will automatically get DNATed to the same IP address as the first packet in that connection.)

Ingress Table 8: Stateful

- For all the configured load balancing rules in **OVN_Northbound** database that includes a L4 port **PORT** of protocol **P** and IPv4 address **VIP**, a priority-120 flow that matches on **ct.new && ip && ip4.dst == VIP && P && P.dst == PORT** with an action of **ct_lb(args)**, where **args** contains comma separated IPv4 addresses (and optional port numbers) to load balance to.

- For all the configured load balancing rules in **OVN_Northbound** database that includes just an IP address *VIP* to match on, a priority-110 flow that matches on **ct.new && ip && ip4.dst == VIP** with an action of **ct_lb(args)**, where *args* contains comma separated IPv4 addresses.
- A priority-100 flow commits packets to connection tracker using **ct_commit; next;** action based on a hint provided by the previous tables (with a match for **reg0[1] == 1**).
- A priority-100 flow sends the packets to connection tracker using **ct_lb;** as the action based on a hint provided by the previous tables (with a match for **reg0[2] == 1**).
- A priority-0 flow that simply moves traffic to the next table.

Ingress Table 9: ARP/ND responder

This table implements ARP/ND responder for known IPs. It contains these logical flows:

- Priority-100 flows to skip ARP responder if inport is of type **localnet**, and advances directly to the next table.
- Priority-50 flows that match ARP requests to each known IP address *A* of every logical router port, and respond with ARP replies directly with corresponding Ethernet address *E*:

```
eth.dst = eth.src;
eth.src = E;
arp.op = 2; /* ARP reply. */
arp.tha = arp.sha;
arp.sha = E;
arp.tpa = arp.spa;
arp.spa = A;
output = inport;
flags.loopback = 1;
output;
```

These flows are omitted for logical ports (other than router ports) that are down.

- Priority-50 flows that match IPv6 ND neighbor solicitations to each known IP address *A* (and *A*'s solicited node address) of every logical router port, and respond with neighbor advertisements directly with corresponding Ethernet address *E*:

```
nd_na {
    eth.src = E;
    ip6.src = A;
    nd.target = A;
    nd.tll = E;
    output = inport;
    flags.loopback = 1;
    output;
};
```

These flows are omitted for logical ports (other than router ports) that are down.

- One priority-0 fallback flow that matches all packets and advances to the next table.

Ingress Table 10: DHCP option processing

This table adds the DHCPv4 options to a DHCPv4 packet from the logical ports configured with IPv4 address(es) and DHCPv4 options.

- A priority-100 logical flow is added for these logical ports which matches the IPv4 packet with **udp.src = 68** and **udp.dst = 67** and applies the action **put_dhcp_opts** and advances the packet to the next table.

```
reg0[3] = put_dhcp_opts(offer_ip = O, options...);
next;
```

For DHCPDISCOVER and DHCPREQUEST, this transforms the packet into a DHCP reply, adds the DHCP offer IP *O* and options to the packet, and stores 1 into reg0[3]. For other kinds of packets, it just stores 0 into reg0[3]. Either way, it continues to the next table.

- A priority-0 flow that matches all packets to advances to table 11.

Ingress Table 11: DHCP responses

This table implements DHCP responder for the DHCP replies generated by the previous table.

- A priority 100 logical flow is added for the logical ports configured with DHCPv4 options which matches IPv4 packets with **udp.src == 68 && udp.dst == 67 && reg0[3] == 1** and responds back to the **inport** after applying these actions. If **reg0[3]** is set to 1, it means that the action **put_dhcp_opts** was successful.

```
eth.dst = eth.src;
eth.src = E;
ip4.dst = O;
ip4.src = S;
udp.src = 67;
udp.dst = 68;
output = P;
flags.loopback = 1;
output;
```

where *E* is the server MAC address and *S* is the server IPv4 address defined in the DHCPv4 options and *O* is the IPv4 address defined in the logical port's addresses column.

(This terminates ingress packet processing; the packet does not go to the next ingress table.)

- A priority-0 flow that matches all packets to advances to table 12.

Ingress Table 12: 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: Pre-LB

This table is similar to ingress table **Pre-LB**. It contains a priority-0 flow that simply moves traffic to the next table. If any load balancing rules exist for the datapath, a priority-100 flow is added with a match of **ip** and action of **reg0[0] = 1; next;** to act as a hint for table **Pre-stateful** to send IP packets to the connection tracker for packet de-fragmentation.

Egress Table 1: to-lport Pre-ACLs

This is similar to ingress table **Pre-ACLs** except for **to-lport** traffic.

Egress Table 2: Pre-stateful

This is similar to ingress table **Pre-stateful**.

Egress Table 3: LB

This is similar to ingress table **LB**.

Egress Table 4: to-lport ACLs

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

Egress Table 5: Stateful

This is similar to ingress table **Stateful** except that there are no rules added for load balancing new connections.

Also a priority 34000 logical flow is added for each logical port which has DHCPv4 options defined to allow the DHCPv4 reply packet from the **Ingress Table 11: DHCP responses**.

Egress Table 6: Egress Port Security - IP

This is similar to the port security logic in table **Ingress Port Security – IP** except that **outputport**, **eth.dst**, **ip4.dst** and **ip6.dst** are checked instead of **inport**, **eth.src**, **ip4.src** and **ip6.src**

Egress Table 7: Egress Port Security - L2

This is similar to the ingress port security logic in ingress table **Admission Control and Ingress Port Security – L2**, but with important differences. Most obviously, **outputport** 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 **outputport** overrides the priority-100 flow with a **drop** action.

Logical Router Datapaths

Logical router datapaths will only exist for **Logical_Router** rows in the **OVN_Northbound** database that do not have **enabled** set to **false**

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-100 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)
 - **ip4.src == 0.0.0.0/8 || ip4.dst == 0.0.0.0/8** (zero network source or destination)
 - **ip4.src** or **ip6.src** is any IP address owned by the router.
 - **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 owned by a router port. Then, for each A that is an IPv4 address, a priority-90 flow matches on **ip4.dst == A and icmp4.type == 8 && icmp4.code == 0** (ICMP echo request). For each A that is an IPv6 address, a priority-90 flow matches on **ip6.dst == A and icmp6.type == 128 && icmp6.code == 0** (ICMPv6 echo request). The port of the router that receives the echo request does not matter. Also, the **ip.ttl** of the echo request packet is not checked, so it complies with RFC 1812, section 4.2.2.9. Flows for ICMPv4 echo requests use the following actions:

```
ip4.dst <=> ip4.src;
ip.ttl = 255;
icmp4.type = 0;
flags.loopback = 1;
next;
```

Flows for ICMPv6 echo requests use the following actions:

```
ip6.dst <=> ip6.src;
ip.ttl = 255;
icmp6.type = 129;
flags.loopback = 1;
next;
```

- Reply to ARP requests. 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-90 flow matches **inport == P && arp.op == 1 && arp.tpa == A** (ARP request) with the following actions:

```
eth.dst = eth.src;
eth.src =  $E$ ;
arp.op = 2; /* ARP reply. */
arp.tha = arp.sha;
arp.sha =  $E$ ;
arp.tpa = arp.spa;
arp.spa =  $A$ ;
outport =  $P$ ;
flags.loopback = 1;
output;
```

- These flows reply to ARP requests for the virtual IP addresses configured in the router for DNAT. For a configured DNAT IP address A , for each router port P with Ethernet address E , a priority-90 flow matches **inport == P && arp.op == 1 && arp.tpa == A** (ARP request) with the following actions:

```
eth.dst = eth.src;
eth.src =  $E$ ;
arp.op = 2; /* ARP reply. */
arp.tha = arp.sha;
arp.sha =  $E$ ;
arp.tpa = arp.spa;
arp.spa =  $A$ ;
outport =  $P$ ;
flags.loopback = 1;
output;
```

- ARP reply handling. This flow uses ARP replies to populate the logical router's ARP table. A priority-90 flow with match **arp.op == 2** has actions **put_arp(inport, arp.spa, arp.sha);**.

- Reply to IPv6 Neighbor Solicitations. These flows reply to Neighbor Solicitation requests for the router's own IPv6 address and populate the logical router's mac binding table. For each router port *P* that owns IPv6 address *A*, solicited node address *S*, and Ethernet address *E*, a priority-90 flow matches **inport == P && nd_ns && ip6.dst == {A, E} && nd.target == A** with the following actions:

```
put_nd(inport, ip6.src, nd.sll);
nd_na {
    eth.src = E;
    ip6.src = A;
    nd.target = A;
    nd.tll = E;
    outport = inport;
    flags.loopback = 1;
    output;
};
```

- IPv6 neighbor advertisement handling. This flow uses neighbor advertisements to populate the logical router's mac binding table. A priority-90 flow with match **nd_na** has actions **put_nd(inport, nd.target, nd.tll);**.
- IPv6 neighbor solicitation for non-hosted addresses handling. This flow uses neighbor solicitations to populate the logical router's mac binding table (ones that were directed at the logical router would have matched the priority-90 neighbor solicitation flow already). A priority-80 flow with match **nd_ns** has actions **put_nd(inport, ip6.src, nd.sll);**.
- UDP port unreachable. Priority-80 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. Not yet implemented.

- TCP reset. Priority-80 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. Not yet implemented.

- Protocol unreachable. Priority-70 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. Not yet implemented.

- 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, which amounts to ICMP (other than echo requests) and fragments with nonzero offsets. For each IP address *A* owned by the router, a priority-60 flow matches **ip4.dst == A** and drops the traffic. An exception is made and the above flow is not added if the router port's own IP address is used to SNAT packets passing through that router.

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-50 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-40 flow with match **inport == *P* && ip.ttl == {0, 1} && !ip.later_frag** matches packets whose TTL has expired, with the following actions to send an ICMP time exceeded reply:

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

Not yet implemented.
- TTL discard. A priority-30 flow with match **ip.ttl == {0, 1}** and actions **drop**; drops other packets whose TTL has expired, that should not receive a ICMP error reply (i.e. fragments with nonzero offset).
- Next table. A priority-0 flows match all packets that aren't already handled and uses actions **next**; to feed them to the ingress table for routing.

Ingress Table 2: UNSNAT

This is for already established connections' reverse traffic. i.e., SNAT has already been done in egress pipeline and now the packet has entered the ingress pipeline as part of a reply. It is unSNATted here.

- For each configuration in the OVN Northbound database, that asks to change the source IP address of a packet from *A* to *B*, a priority-100 flow matches **ip && ip4.dst == *B*** with an action **ct_snat; next**;
A priority-0 logical flow with match **1** has actions **next**;

Ingress Table 3: DNAT

Packets enter the pipeline with destination IP address that needs to be DNATted from a virtual IP address to a real IP address. Packets in the reverse direction needs to be unDNATed.

- For each configuration in the OVN Northbound database, that asks to change the destination IP address of a packet from *A* to *B*, a priority-100 flow matches **ip && ip4.dst == *A*** with an action **flags.loopback = 1; ct_dnat(*B*)**;
For all IP packets of a Gateway router, a priority-50 flow with an action **flags.loopback = 1; ct_dnat**;
A priority-0 logical flow with match **1** has actions **next**;

Ingress Table 4: IP Routing

A packet that arrives at this table is an IP packet that should be routed to the address in **ip4.dst** or **ip6.dst**. This table implements IP routing, setting **reg0** (or **xxreg0** for IPv6) to the next-hop IP address (leaving **ip4.dst** or **ip6.dst**, the packet's final destination, unchanged) and advances to the next table for ARP resolution. It also sets **reg1** (or **xxreg1** to the IP address owned by the selected router port (which is used later in table 6 as the IP source address for an ARP request, if needed).

This table contains the following logical flows:

- IPv4 routing table. For each route to IPv4 network *N* with netmask *M*, on router port *P* with 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:

```
ip.ttl--;
reg0 = G;
reg1 = A;
eth.src = E;
```

```

output = P;
flags.loopback = 1;
next;

```

(Ingress table 1 already verified that **ip.ttl**—; will not yield a TTL exceeded error.)

If the route has a gateway, *G* is the gateway IP address. Instead, if the route is from a configured static route, *G* is the next hop IP address. Else it is **ip4.dst**.

- IPv6 routing table. For each route to IPv6 network *N* with netmask *M*, on router port *P* with IP address *A* and Ethernet address *E*, a logical flow with match in CIDR notation **ip6.dst** == *N/M*, whose priority is the integer value of *M*, has the following actions:

```

ip.ttl—;
xxreg0 = G;
xxreg1 = A;
eth.src = E;
output = P;
flags.loopback = 1;
next;

```

(Ingress table 1 already verified that **ip.ttl**—; will not yield a TTL exceeded error.)

If the route has a gateway, *G* is the gateway IP address. Instead, if the route is from a configured static route, *G* is the next hop IP address. Else it is **ip6.dst**.

If the address *A* is in the link-local scope, the route will be limited to sending on the ingress port.

Ingress Table 5: ARP/ND Resolution

Any packet that reaches this table is an IP packet whose next-hop IPv4 address is in **reg0** or IPv6 address is in **xxreg0**. (**ip4.dst** or **ip6.dst** contains the final destination.) This table resolves the IP address in **reg0** (or **xxreg0**) into an output port in **output** and an Ethernet address in **eth.dst**, using the following flows:

- Static MAC bindings. 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_Switch_Port** table. For router ports connected to other logical routers, MAC bindings can be known statically from the **mac** and **networks** column in the **Logical_Router_Port** table.

For each IPv4 address *A* whose host is known to have Ethernet address *E* on router port *P*, a priority-100 flow with match **output** === *P* && **reg0** == *A* has actions **eth.dst** = *E*; **next**;

For each IPv6 address *A* whose host is known to have Ethernet address *E* on router port *P*, a priority-100 flow with match **output** === *P* && **xxreg0** == *A* has actions **eth.dst** = *E*; **next**;

For each logical router port with an IPv4 address *A* and a mac address of *E* that is reachable via a different logical router port *P*, a priority-100 flow with match **output** === *P* && **reg0** == *A* has actions **eth.dst** = *E*; **next**;

For each logical router port with an IPv6 address *A* and a mac address of *E* that is reachable via a different logical router port *P*, a priority-100 flow with match **output** === *P* && **xxreg0** == *A* has actions **eth.dst** = *E*; **next**;

- Dynamic MAC bindings. These flows resolve MAC-to-IP bindings that have become known dynamically through ARP or neighbor discovery. (The next table will issue an ARP or neighbor solicitation request for cases where the binding is not yet known.)

A priority-0 logical flow with match **ip4** has actions **get_arp(output, reg0)**; **next**;

A priority-0 logical flow with match **ip6** has actions **get_nd(output, xxreg0)**; **next**;

Ingress Table 6: ARP Request

In the common case where the Ethernet destination has been resolved, this table outputs the packet. Otherwise, it composes and sends an ARP request. It holds the following flows:

- Unknown MAC address. A priority-100 flow with match **eth.dst == 00:00:00:00:00:00** has the following actions:

```
arp {
    eth.dst = ff:ff:ff:ff:ff:ff;
    arp.spa = reg1;
    arp.op = 1; /* ARP request. */
    output;
};
```

(Ingress table 4 initialized **reg1** with the IP address owned by **outport**.)

The IP packet that triggers the ARP request is dropped.

- Known MAC address. A priority-0 flow with match **1** has actions **output;**.

Egress Table 0: SNAT

Packets that are configured to be SNATed get their source IP address changed based on the configuration in the OVN Northbound database.

- For each configuration in the OVN Northbound database, that asks to change the source IP address of a packet from an IP address of *A* or to change the source IP address of a packet that belongs to network *A* to *B*, a flow matches **ip && ip4.src == A** with an action **ct_snat(B)**. The priority of the flow is calculated based on the mask of *A*, with matches having larger masks getting higher priorities.

A priority-0 logical flow with match **1** has actions **next;**.

Egress Table 1: 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;**.