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.

OPTIONS

−−ovnnb−db=*database*

The OVSDB database containing the OVN Northbound Database. If the **OVN_NB_DB** environment variable is set, its value is used as the default. Otherwise, the default is **unix:/var/run/openvswitch/ovnnb_db.sock**.

−−ovnsb−db=*database*

The OVSDB database containing the OVN Southbound Database. If the **OVN_SB_DB** environment variable is set, its value is used as the default. Otherwise, the default is **unix:/var/run/openvswitch/ovnsb_db.sock**.

database in the above options must be an OVSDB active or passive connection method, as described in **ovsdb**(7).

Daemon Options

−−pidfile[**=***pidfile*]

Causes a file (by default, *program***.pid**) to be created indicating the PID of the running process. If the *pidfile* argument is not specified, or if it does not begin with **/**, then it is created in **/var/run/openvswitch**.

If **−−pidfile** is not specified, no pidfile is created.

−−overwrite−pidfile

By default, when **−−pidfile** is specified and the specified pidfile already exists and is locked by a running process, the daemon refuses to start. Specify **−−overwrite−pidfile** to cause it to instead overwrite the pidfile.

When **−−pidfile** is not specified, this option has no effect.

−−detach

Runs this program as a background process. The process forks, and in the child it starts a new session, closes the standard file descriptors (which has the side effect of disabling logging to the console), and changes its current directory to the root (unless **−−no−chdir** is specified). After the child completes its initialization, the parent exits.

−−monitor

Creates an additional process to monitor this program. If it dies due to a signal that indicates a programming error (**SIGABRT**, **SIGALRM**, **SIGBUS**, **SIGFPE**, **SIGILL**, **SIGPIPE**, **SIGSEGV**, **SIGXCPU**, or **SIGXFSZ**) then the monitor process starts a new copy of it. If the daemon dies or exits for another reason, the monitor process exits.

This option is normally used with **−−detach**, but it also functions without it.

−−no−chdir

By default, when **−−detach** is specified, the daemon changes its current working directory to the root directory after it detaches. Otherwise, invoking the daemon from a carelessly chosen directory would prevent the administrator from unmounting the file system that holds that directory.

Specifying **−−no−chdir** suppresses this behavior, preventing the daemon from changing its current working directory. This may be useful for collecting core files, since it is common behavior to write core dumps into the current working directory and the root directory is not a good directory

to use.

This option has no effect when **−−detach** is not specified.

−−no−self−confinement
By default this daemon will try to self-confine itself to work with files under well-known directories whitelisted at build time. It is better to stick with this default behavior and not to use this flag unless some other Access Control is used to confine daemon. Note that in contrast to other access control implementations that are typically enforced from kernel-space (e.g. DAC or MAC), self-confinement is imposed from the user-space daemon itself and hence should not be considered as a full confinement strategy, but instead should be viewed as an additional layer of security.

−−user=*user***:***group*
Causes this program to run as a different user specified in *user***:***group*, thus dropping most of the root privileges. Short forms *user* and **:***group* are also allowed, with current user or group assumed, respectively. Only daemons started by the root user accepts this argument.

On Linux, daemons will be granted **CAP_IPC_LOCK** and **CAP_NET_BIND_SERVICES** before dropping root privileges. Daemons that interact with a datapath, such as **ovs−vswitchd**, will be granted three additional capabilities, namely **CAP_NET_ADMIN**, **CAP_NET_BROADCAST** and **CAP_NET_RAW**. The capability change will apply even if the new user is root.

On Windows, this option is not currently supported. For security reasons, specifying this option will cause the daemon process not to start.

Logging Options

−v[*spec*]
−−verbose=[*spec*]
Sets logging levels. Without any *spec*, sets the log level for every module and destination to **dbg**. Otherwise, *spec* is a list of words separated by spaces or commas or colons, up to one from each category below:

- A valid module name, as displayed by the **vlog/list** command on **ovs−appctl**(8), limits the log level change to the specified module.
- **syslog**, **console**, or **file**, to limit the log level change to only to the system log, to the console, or to a file, respectively. (If **−−detach** is specified, the daemon closes its standard file descriptors, so logging to the console will have no effect.)

  On Windows platform, **syslog** is accepted as a word and is only useful along with the **−−syslog−target** option (the word has no effect otherwise).
- **off**, **emer**, **err**, **warn**, **info**, or **dbg**, to control the log level. Messages of the given severity or higher will be logged, and messages of lower severity will be filtered out. **off** filters out all messages. See **ovs−appctl**(8) for a definition of each log level.

Case is not significant within *spec*.

Regardless of the log levels set for **file**, logging to a file will not take place unless **−−log−file** is also specified (see below).

For compatibility with older versions of OVS, **any** is accepted as a word but has no effect.

−v
−−verbose
Sets the maximum logging verbosity level, equivalent to **−−verbose=dbg**.

−vPATTERN:*destination***:***pattern*
−−verbose=PATTERN:*destination***:***pattern*
Sets the log pattern for *destination* to *pattern*. Refer to **ovs−appctl**(8) for a description of the valid syntax for *pattern*.

−vFACILITY:*facility*
−−verbose=FACILITY:*facility*
Sets the RFC5424 facility of the log message. *facility* can be one of **kern**, **user**, **mail**, **daemon**, **auth**, **syslog**, **lpr**, **news**, **uucp**, **clock**, **ftp**, **ntp**, **audit**, **alert**, **clock2**, **local0**, **local1**, **local2**, **local3**, **local4**, **local5**, **local6** or **local7**. If this option is not specified, **daemon** is used as the default for the local system syslog and **local0** is used while sending a message to the target provided via the **−−syslog−target** option.

−−log−file[**=***file*]
Enables logging to a file. If *file* is specified, then it is used as the exact name for the log file. The default log file name used if *file* is omitted is **/var/log/openvswitch/***program***.log**.

−−syslog−target=*host***:***port*
Send syslog messages to UDP *port* on *host*, in addition to the system syslog. The *host* must be a numerical IP address, not a hostname.

−−syslog−method=*method*
Specify *method* as how syslog messages should be sent to syslog daemon. The following forms are supported:

- **libc**, to use the libc **syslog()** function. Downside of using this options is that libc adds fixed prefix to every message before it is actually sent to the syslog daemon over **/dev/log** UNIX domain socket.
- **unix:***file*, to use a UNIX domain socket directly. It is possible to specify arbitrary message format with this option. However, **rsyslogd 8.9** and older versions use hard coded parser function anyway that limits UNIX domain socket use. If you want to use arbitrary message format with older **rsyslogd** versions, then use UDP socket to localhost IP address instead.
- **udp:***ip***:***port*, to use a UDP socket. With this method it is possible to use arbitrary message format also with older **rsyslogd**. When sending syslog messages over UDP socket extra precaution needs to be taken into account, for example, syslog daemon needs to be configured to listen on the specified UDP port, accidental iptables rules could be interfering with local syslog traffic and there are some security considerations that apply to UDP sockets, but do not apply to UNIX domain sockets.
- **null**, to discard all messages logged to syslog.

The default is taken from the **OVS_SYSLOG_METHOD** environment variable; if it is unset, the default is **libc**.

PKI Options
PKI configuration is required in order to use SSL for the connections to the Northbound and Southbound databases.

−p *privkey.pem*
−−private−key=*privkey.pem*
Specifies a PEM file containing the private key used as identity for outgoing SSL connections.

−c *cert.pem*
−−certificate=*cert.pem*
Specifies a PEM file containing a certificate that certifies the private key specified on **−p** or **−−private−key** to be trustworthy. The certificate must be signed by the certificate authority (CA) that the peer in SSL connections will use to verify it.

−C *cacert.pem*
−−ca−cert=*cacert.pem*
Specifies a PEM file containing the CA certificate for verifying certificates presented to this program by SSL peers. (This may be the same certificate that SSL peers use to verify the certificate specified on **−c** or **−−certificate**, or it may be a different one, depending on

the PKI design in use.)

−C none
−−ca−cert=none
Disables verification of certificates presented by SSL peers. This introduces a security risk, because it means that certificates cannot be verified to be those of known trusted hosts.

Other Options

−−unixctl=*socket*
Sets the name of the control socket on which *program* listens for runtime management commands (see *RUNTIME MANAGEMENT COMMANDS,* below). If *socket* does not begin with **/**, it is interpreted as relative to **/var/run/openvswitch**. If **−−unixctl** is not used at all, the default socket is **/var/run/openvswitch/***program***.***pid***.ctl**, where *pid* is *program*'s process ID.

On Windows a local named pipe is used to listen for runtime management commands. A file is created in the absolute path as pointed by *socket* or if **−−unixctl** is not used at all, a file is created as *program* in the configured *OVS_RUNDIR* directory. The file exists just to mimic the behavior of a Unix domain socket.

Specifying **none** for *socket* disables the control socket feature.

−h
−−help Prints a brief help message to the console.

−V
−−version
Prints version information to the console.

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.

ACTIVE-STANDBY FOR HIGH AVAILABILITY

You may run **ovn−northd** more than once in an OVN deployment. OVN will automatically ensure that only one of them is active at a time. If multiple instances of **ovn−northd** are running and the active **ovn−northd** fails, one of the hot standby instances of **ovn−northd** will automatically take over.

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 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**. If special ports such as route ports or localnet ports can't use ct(), a priority−110 flow is added to skip over stateful 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. Moreover it contains a priority−110 flow to move IPv6 Neighbor Discovery traffic to the next table. If load balancing rules with virtual IP addresses (and ports) are configured in **OVN_Northbound** database for a logical switch datapath, a priority−100 flow is added for each configured virtual IP address *VIP*. For IPv4 *VIPs*, the match is **ip && ip4.dst ==** *VIP*. For IPv6 *VIPs*, the match is **ip && ip6.dst ==** *VIP*. The flow 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 statetful 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.blocked** 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.blocked** 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.blocked** set.
- A priority−65535 flow that drops all traffic marked by the connection tracker as invalid.
- A priority−65535 flow that drops all trafic in the reply direction with **ct_label.blocked** 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: **from−lport** *QoS Marking*

Logical flows in this table closely reproduce those in the **QoS** table with the **action** column set in the **OVN_Northbound** database for the **from−lport** direction.

- For every qos_rules entry in a logical switch with DSCP marking enabled, a flow will be added at the priority mentioned in the QoS table.
- One priority−0 fallback flow that matches all packets and advances to the next table.

Ingress Table 8: **from−lport** *QoS Meter*

Logical flows in this table closely reproduce those in the **QoS** table with the **bandwidth** column set in the **OVN_Northbound** database for the **from−lport** direction.

- For every qos_rules entry in a logical switch with metering enabled, a flow will be added at the priorirty mentioned in the QoS table.

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

Ingress Table 9: 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 10: Stateful

- For all the configured load balancing rules for a switch in **OVN_Northbound** database that includes a L4 port *PORT* of protocol *P* and IP address *VIP*, a priority−120 flow is added. For IPv4 *VIPs* , the flow matches **ct.new && ip && ip4.dst ==** *VIP* **&&** *P* **&& *P*.dst ==** *PORT*. For IPv6 *VIPs*, the flow matches **ct.new && ip && ip6.dst ==** *VIP* **&&** *P* **&& *P*.dst ==** *PORT*. The flow's action is **ct_lb(***args***)** , where *args* contains comma separated IP addresses (and optional port numbers) to load balance to. The address family of the IP addresses of *args* is the same as the address family of *VIP*
- For all the configured load balancing rules for a switch in **OVN_Northbound** database that includes just an IP address *VIP* to match on, OVN adds a priority−110 flow. For IPv4 *VIPs*, the flow matches **ct.new && ip && ip4.dst ==** *VIP*. For IPv6 *VIPs*, the flow matches **ct.new && ip && ip6.dst ==** *VIP*. The action on this flow is **ct_lb(***args***)**, where *args* contains comma separated IP addresses of the same address family as *VIP*.
- 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 11: ARP/ND responder

This table implements ARP/ND responder in a logical switch for known IPs. The advantage of the ARP responder flow is to limit ARP broadcasts by locally responding to ARP requests without the need to send to other hypervisors. One common case is when the inport is a logical port associated with a VIF and the broadcast is responded to on the local hypervisor rather than broadcast across the whole network and responded to by the destination VM. This behavior is proxy ARP.

ARP requests arrive from VMs from a logical switch inport of type default. For this case, the logical switch proxy ARP rules can be for other VMs or logical router ports. Logical switch proxy ARP rules may be programmed both for mac binding of IP addresses on other logical switch VIF ports (which are of the default logical switch port type, representing connectivity to VMs or containers), and for mac binding of IP addresses on logical switch router type ports, representing their logical router port peers. In order to support proxy ARP for logical router ports, an IP address must be configured on the logical switch router type port, with the same value as the peer logical router port. The configured MAC addresses must match as well. When a VM sends an ARP request for a distributed logical router port and if the peer router type port of the attached logical switch does not have an IP address configured, the ARP request will be broadcast on the logical switch. One of the copies of the ARP request will go through the logical switch router type port to the logical router datapath, where the logical router ARP responder will generate a reply. The MAC binding of a distributed logical router, once learned by an associated VM, is used for all that VM's communication needing routing. Hence, the action of a VM re-arping for the mac binding of the logical router port should be rare.

Logical switch ARP responder proxy ARP rules can also be hit when receiving ARP requests externally on a L2 gateway port. In this case, the hypervisor acting as an L2 gateway, responds to the ARP request on behalf of a destination VM.

Note that ARP requests received from **localnet** or **vtep** logical inports can either go directly to VMs, in which case the VM responds or can hit an ARP responder for a logical router port if the packet is used to

resolve a logical router port next hop address. In either case, logical switch ARP responder rules will not be hit. It contains these logical flows:

- Priority−100 flows to skip the ARP responder if inport is of type **localnet** or **vtep** and advances directly to the next table. ARP requests sent to **localnet** or **vtep** ports can be received by multiple hypervisors. Now, because the same mac binding rules are downloaded to all hypervisors, each of the multiple hypervisors will respond. This will confuse L2 learning on the source of the ARP requests. ARP requests received on an inport of type **router** are not expected to hit any logical switch ARP responder flows. However, no skip flows are installed for these packets, as there would be some additional flow cost for this and the value appears limited.

- Priority−50 flows that match ARP requests to each known IP address *A* of every logical switch 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*;
  outport = inport;
  flags.loopback = 1;
  output;

  These flows are omitted for logical ports (other than router ports or **localport** 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 switch port except of type router, 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*;
  outport = inport;
  flags.loopback = 1;
  output;
  };

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

  nd_na_router {
  eth.src = *E*;
  ip6.src = *A*;
  nd.target = *A*;
  nd.tll = *E*;
  outport = inport;
  flags.loopback = 1;
  output;
  };

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

- Priority−100 flows with match criteria like the ARP and ND flows above, except that they only match packets from the **inport** that owns the IP addresses in question, with action **next;**. These flows prevent OVN from replying to, for example, an ARP request emitted by a VM for its own IP address. A VM only makes this kind of request to attempt to detect a duplicate IP address assignment, so sending a reply will prevent the VM from accepting the IP address that it owns.

  In place of **next;**, it would be reasonable to use **drop;** for the flows' actions. If everything is working as it is configured, then this would produce equivalent results, since no host should reply to the request. But ARPing for one's own IP address is intended to detect situations where the network is not working as configured, so dropping the request would frustrate that intent.

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

Ingress Table 12: 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, and similarly for DHCPv6 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 = *ip*, *options...*);
  next;

  For DHCPDISCOVER and DHCPREQUEST, this transforms the packet into a DHCP reply, adds the DHCP offer IP *ip* 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−100 logical flow is added for these logical ports which matches the IPv6 packet with **udp.src** = 546 and **udp.dst** = 547 and applies the action **put_dhcpv6_opts** and advances the packet to the next table.

  reg0[3] = put_dhcpv6_opts(ia_addr = *ip*, *options...*);
  next;

  For DHCPv6 Solicit/Request/Confirm packets, this transforms the packet into a DHCPv6 Advertise/Reply, adds the DHCPv6 offer IP *ip* 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 13: 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 = *A*;
  ip4.src = *S*;

udp.src = 67;
udp.dst = 68;
outport = *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 *A* 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 100 logical flow is added for the logical ports configured with DHCPv6 options which matches IPv6 packets with **udp.src == 546 && udp.dst == 547 && 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_dhcpv6_opts** was successful.

  eth.dst = eth.src;
  eth.src = *E*;
  ip6.dst = *A*;
  ip6.src = *S*;
  udp.src = 547;
  udp.dst = 546;
  outport = *P*;
  flags.loopback = 1;
  output;

  where *E* is the server MAC address and *S* is the server IPv6 LLA address generated from the **server_id** defined in the DHCPv6 options and *A* is the IPv6 address defined in the logical port's addresses column.

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

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

Ingress Table 14 DNS Lookup

This table looks up and resolves the DNS names to the corresponding configured IP address(es).

- A priority−100 logical flow for each logical switch datapath if it is configured with DNS records, which matches the IPv4 and IPv6 packets with **udp.dst** = 53 and applies the action **dns_lookup** and advances the packet to the next table.

  reg0[4] = dns_lookup(); next;

  For valid DNS packets, this transforms the packet into a DNS reply if the DNS name can be resolved, and stores 1 into reg0[4]. For failed DNS resolution or other kinds of packets, it just stores 0 into reg0[4]. Either way, it continues to the next table.

Ingress Table 15 DNS Responses

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

- A priority−100 logical flow for each logical switch datapath if it is configured with DNS records, which matches the IPv4 and IPv6 packets with **udp.dst = 53 && reg0[4] == 1** and responds back to the **inport** after applying these actions. If **reg0[4]** is set to 1, it means that the action **dns_lookup** was successful.

  eth.dst <−> eth.src;
  ip4.src <−> ip4.dst;
  udp.dst = udp.src;

udp.src = 53;
outport = *P*;
flags.loopback = 1;
output;

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

Ingress Table 16 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.

  For the Ethernet address on a logical switch port of type **router**, when that logical switch port's **addresses** column is set to **router** and the connected logical router port specifies a **redirect−chassis**:

  - The flow for the connected logical router port's Ethernet address is only programmed on the **redirect−chassis**.
  - If the logical router has rules specified in **nat** with **external_mac**, then those addresses are also used to populate the switch's destination lookup on the chassis where **logical_port** is resident.

  For the Ethernet address on a logical switch port of type **router**, when that logical switch port's **addresses** column is set to **router** and the connected logical router port specifies a **reside−on−redirect−chassis** and the logical router to which the connected logical router port belongs to has a **redirect−chassis** distributed gateway logical router port:

  - The flow for the connected logical router port's Ethernet address is only programmed on the **redirect−chassis**.
- 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. Moreover it contains a priority−110 flow to move IPv6 Neighbor Discovery 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.

In addition, the following flows are added.

- A priority 34000 logical flow is added for each logical port which has DHCPv4 options defined to allow the DHCPv4 reply packet and which has DHCPv6 options defined to allow the DHCPv6 reply packet from the **Ingress Table 13: DHCP responses**.
- A priority 34000 logical flow is added for each logical switch datapath configured with DNS records with the match **udp.dst = 53** to allow the DNS reply packet from the **Ingress Table 15:DNS responses**.

Egress Table 5: **to−lport** *QoS Marking*

This is similar to ingress table **QoS marking** except they apply to **to−lport** QoS rules.

Egress Table 6: **to−lport** *QoS Meter*

This is similar to ingress table **QoS meter** except they apply to **to−lport** QoS rules.

Egress Table 7: Stateful

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

Egress Table 8: Egress Port Security - IP

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

Egress Table 9: 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, **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

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;**.

  For the gateway port on a distributed logical router (where one of the logical router ports specifies a **redirect−chassis**), the above flow matching **eth.dst ==** *E* is only programmed on the gateway port instance on the **redirect−chassis**.
- For each **dnat_and_snat** NAT rule on a distributed router that specifies an external Ethernet address *E*, a priority−50 flow that matches **inport ==** *GW* **&& eth.dst ==** *E*, where *GW* is the logical router gateway port, with action **next;**.

  This flow is only programmed on the gateway port instance on the chassis where the **logical_port** specified in the NAT rule resides.

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, unless the packet was recirculated due to egress loopback as indicated by **REGBIT_EGRESS_LOOPBACK**.
  - **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 and populates mac binding table of the logical router port. The ARP requests are handled only if the requestor's IP belongs to the same subnets of the logical router port. For each router port *P* that owns IP address *A*, which belongs to subnet *S* with prefix length *L*, and Ethernet address *E*, a priority−90 flow matches **inport ==** *P* **&& arp.spa ==** *S/L* **&& arp.op == 1 && arp.tpa ==** *A* (ARP request) with the following actions:

  put_arp(inport, arp.spa, arp.sha);
  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;

For the gateway port on a distributed logical router (where one of the logical router ports specifies a **redirect−chassis**), the above flows are only programmed on the gateway port instance on the **redirect−chassis**. This behavior avoids generation of multiple ARP responses from different chassis, and allows upstream MAC learning to point to the **redirect−chassis**.

For the logical router port with the option **reside−on−redirect−chassis** set (which is centralized), the above flows are only programmed on the gateway port instance on the **redirect−chassis** (if the logical router has a distributed gateway port). This behavior avoids generation of multiple ARP responses from different chassis, and allows upstream MAC learning to point to the **redirect−chassis**.

- These flows handles ARP requests not for router's own IP address. They use the SPA and SHA to populate the logical router port's mac binding table, with priority 80. The typical use case of these flows are GARP requests handling. For the gateway port on a distributed logical router, these flows are only programmed on the gateway port instance on the **redirect−chassis**.

- These flows reply to ARP requests for the virtual IP addresses configured in the router for DNAT or load balancing. For a configured DNAT IP address or a load balancer IPv4 VIP *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;

  For the gateway port on a distributed logical router with NAT (where one of the logical router ports specifies a **redirect−chassis**):

  - If the corresponding NAT rule cannot be handled in a distributed manner, then this flow is only programmed on the gateway port instance on the **redirect−chassis**. This behavior avoids generation of multiple ARP responses from different chassis, and allows upstream MAC learning to point to the **redirect−chassis**.

  - If the corresponding NAT rule can be handled in a distributed manner, then this flow is only programmed on the gateway port instance where the **logical_port** specified in the NAT rule resides.

    Some of the actions are different for this case, using the **external_mac** specified in the NAT rule rather than the gateway port's Ethernet address *E*:

    eth.src = *external_mac*;
    arp.sha = *external_mac*;

This behavior avoids generation of multiple ARP responses from different chassis, and allows upstream MAC learning to point to the correct chassis.

- 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 load balancing IPv6 VIPs 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_router {
  eth.src = *E*;
  ip6.src = *A*;
  nd.target = *A*;
  nd.tll = *E*;
  outport = inport;
  flags.loopback = 1;
  output;
  };

  For each router port *P* that has load balancing VIP *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;
  };

  For the gateway port on a distributed logical router (where one of the logical router ports specifies a **redirect−chassis**), the above flows replying to IPv6 Neighbor Solicitations are only programmed on the gateway port instance on the **redirect−chassis**. This behavior avoids generation of multiple replies from different chassis, and allows upstream MAC learning to point to the **redirect−chassis**.
- 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, except in the special case of

gateways, which accept traffic directed to a router IP for load balancing and NAT purposes.

These flows should not match IP fragments with nonzero offset.

- TCP reset. Priority−80 flows generate TCP reset messages in reply to TCP datagrams directed to the router's IP address, except in the special case of gateways, which accept traffic directed to a router IP for load balancing and NAT purposes.

  These flows should not match IP fragments with nonzero offset.

- Protocol or address unreachable. Priority−70 flows generate ICMP protocol or address unreachable messages for IPv4 and IPv6 respectively in reply to packets directed to the router's IP address on IP protocols other than UDP, TCP, and ICMP, except in the special case of gateways, which accept traffic directed to a router IP for load balancing purposes.

  These flows should not match IP fragments with nonzero offset.

- 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 for IPv4 and IPv6 respectively:

  ```
  icmp4 {
    icmp4.type = 11; /* Time exceeded. */
    icmp4.code = 0;  /* TTL exceeded in transit. */
    ip4.dst = ip4.src;
    ip4.src = A;
    ip.ttl = 255;
    next;
  };
  icmp6 {
    icmp6.type = 3; /* Time exceeded. */
    icmp6.code = 0;  /* TTL exceeded in transit. */
    ip6.dst = ip6.src;
    ip6.src = A;
    ip.ttl = 255;
    next;
  };
  ```

- 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 next table.

Ingress Table 2: DEFRAG

This is to send packets to connection tracker for tracking and defragmentation. 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 Gateway router, a priority−100 flow is added for each configured virtual IP address *VIP*. For IPv4 *VIPs* the flow matches **ip && ip4.dst ==** *VIP*. For IPv6 *VIPs*, the flow matches **ip && ip6.dst ==** *VIP*. The flow uses the action **ct_next;** to send IP packets to the connection tracker for packet de-fragmentation and tracking before sending it to the next table.

Ingress Table 3: 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.

Ingress Table 3: UNSNAT on Gateway Routers

- If the Gateway router has been configured to force SNAT any previously DNATted packets to *B*, a priority−110 flow matches **ip && ip4.dst ==** *B* with an action **ct_snat;** .

  If the Gateway router has been configured to force SNAT any previously load-balanced packets to *B*, a priority−100 flow matches **ip && ip4.dst ==** *B* with an action **ct_snat;** .

  For each NAT configuration in the OVN Northbound database, that asks to change the source IP address of a packet from *A* to *B*, a priority−90 flow matches **ip && ip4.dst ==** *B* with an action **ct_snat;** .

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

Ingress Table 3: UNSNAT on Distributed Routers

- 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* **&& inport ==** *GW*, where *GW* is the logical router gateway port, with an action **ct_snat;**.

  If the NAT rule cannot be handled in a distributed manner, then the priority−100 flow above is only programmed on the **redirect−chassis**.

  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−50 flow matches **ip && ip4.dst ==** *B* with an action **REGBIT_NAT_REDIRECT = 1; next;**. This flow is for east/west traffic to a NAT destination IPv4 address. By setting the **REGBIT_NAT_REDIRECT** flag, in the ingress table **Gateway Redirect** this will trigger a redirect to the instance of the gateway port on the **redirect−chassis**.

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

Ingress Table 4: 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.

Ingress Table 4: Load balancing DNAT rules

Following load balancing DNAT flows are added for Gateway router or Router with gateway port. These flows are programmed only on the **redirect−chassis**. These flows do not get programmed for load balancers with IPv6 *VIPs*.

- For all the configured load balancing rules for a Gateway router or Router with gateway port 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. If the router is configured to force SNAT any load-balanced packets, the above action will be replaced by **flags.force_snat_for_lb = 1; ct_lb(***args***);**.

- For all the configured load balancing rules for a router 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.est && ip && ip4.dst ==** *VIP* **&&** *P* **&& *P*.dst ==** *PORT*
  with an action of **ct_dnat;**. If the router is configured to force SNAT any load-balanced packets, the above action will be replaced by **flags.force_snat_for_lb = 1; ct_dnat;**.
- For all the configured load balancing rules for a router 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. If the router is configured to force SNAT any load-balanced packets, the above action will be replaced by **flags.force_snat_for_lb = 1; ct_lb(***args***);**.
- For all the configured load balancing rules for a router in **OVN_Northbound** database that includes just an IP address *VIP* to match on, a priority−110 flow that matches on **ct.est && ip && ip4.dst ==** *VIP* with an action of **ct_dnat;**. If the router is configured to force SNAT any load-balanced packets, the above action will be replaced by **flags.force_snat_for_lb = 1; ct_dnat;**.

Ingress Table 4: DNAT on Gateway Routers

- 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***);**. If the Gateway router is configured to force SNAT any DNATed packet, the above action will be replaced by **flags.force_snat_for_dnat = 1; 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: DNAT on Distributed Routers

On distributed routers, the DNAT table only handles packets with destination IP address that needs to be DNATted from a virtual IP address to a real IP address. The unDNAT processing in the reverse direction is handled in a separate table in the egress pipeline.

- 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 ==** *B* **&& inport ==** *GW*, where *GW* is the logical router gateway port, with an action **ct_dnat(***B***);**.

  If the NAT rule cannot be handled in a distributed manner, then the priority−100 flow above is only programmed on the **redirect−chassis**.

  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−50 flow matches **ip && ip4.dst ==** *B* with an action **REGBIT_NAT_REDIRECT = 1; next;**. This flow is for east/west traffic to a NAT destination IPv4 address. By setting the **REGBIT_NAT_REDIRECT** flag, in the ingress table **Gateway Redirect** this will trigger a redirect to the instance of the gateway port on the **redirect−chassis**.

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

Ingress Table 5: IPv6 ND RA option processing

- A priority−50 logical flow is added for each logical router port configured with IPv6 ND RA options which matches IPv6 ND Router Solicitation packet and applies the action **put_nd_ra_opts** and advances the packet to the next table.

  reg0[5] = put_nd_ra_opts(*options***)**;next;

For a valid IPv6 ND RS packet, this transforms the packet into an IPv6 ND RA reply and sets the RA options to the packet and stores 1 into reg0[5]. For other kinds of packets, it just stores 0 into reg0[5]. Either way, it continues to the next table.

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

Ingress Table 6: IPv6 ND RA responder

This table implements IPv6 ND RA responder for the IPv6 ND RA replies generated by the previous table.

- A priority−50 logical flow is added for each logical router port configured with IPv6 ND RA options which matches IPv6 ND RA packets and **reg0[5] == 1** and responds back to the **inport** after applying these actions. If **reg0[5]** is set to 1, it means that the action **put_nd_ra_opts** was successful.

  eth.dst = eth.src;
  eth.src = *E*;
  ip6.dst = ip6.src;
  ip6.src = *I*;
  outport = *P*;
  flags.loopback = 1;
  output;

  where *E* is the MAC address and *I* is the IPv6 link local address of the logical router port.

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

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

Ingress Table 7: 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 (ingress table **ARP Request** will generate an ARP request, if needed, with **reg0** as the target protocol address and **reg1** as the source protocol address).

This table contains the following logical flows:

- For distributed logical routers where one of the logical router ports specifies a **redirect−chassis**, a priority−300 logical flow with match **REGBIT_NAT_REDIRECT == 1** has actions **ip.ttl−−; next;**. The **outport** will be set later in the Gateway Redirect table.

- 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*;
  outport = *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*;
  outport = *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 8: 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 **outport** and an Ethernet address in **eth.dst**, using the following flows:

- For distributed logical routers where one of the logical router ports specifies a **redirect−chassis**, a priority−200 logical flow with match **REGBIT_NAT_REDIRECT == 1** has actions **eth.dst =** *E***; next;**, where *E* is the ethernet address of the router's distributed gateway port.
- 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 **outport ===** *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 **outport ===** *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 **outport ===** *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 **outport ===** *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 ingress table **ARP Request** 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(outport, reg0); next;**.

  A priority−0 logical flow with match **ip6** has actions **get_nd(outport, xxreg0); next;**.

Ingress Table 9: Gateway Redirect

For distributed logical routers where one of the logical router ports specifies a **redirect−chassis**, this table redirects certain packets to the distributed gateway port instance on the **redirect−chassis**. This table has the following flows:

- A priority−200 logical flow with match **REGBIT_NAT_REDIRECT == 1** has actions **outport =** *CR***; next;**, where *CR* is the **chassisredirect** port representing the instance of the logical router distributed gateway port on the **redirect−chassis**.
- A priority−150 logical flow with match **outport ==** *GW* **&& eth.dst == 00:00:00:00:00:00** has actions **outport =** *CR***; next;**, where *GW* is the logical router distributed gateway port and *CR* is the **chassisredirect** port representing the instance of the logical router distributed gateway port on the **redirect−chassis**.
- For each NAT rule in the OVN Northbound database that can be handled in a distributed manner, a priority−100 logical flow with match **ip4.src ==** *B* **&& outport ==** *GW*, where *GW* is the logical router distributed gateway port, with actions **next;**.
- A priority−50 logical flow with match **outport ==** *GW* has actions **outport =** *CR***; next;**, where *GW* is the logical router distributed gateway port and *CR* is the **chassisredirect** port representing the instance of the logical router distributed gateway port on the **redirect−chassis**.
- A priority−0 logical flow with match **1** has actions **next;**.

Ingress Table 10: 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 or IPv6 Neighbor Solicitation request. It holds the following flows:

- Unknown MAC address. A priority−100 flow for IPv4 packets 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.tpa = reg0;
    arp.op = 1;  /* ARP request. */
    output;
  };
  ```

  Unknown MAC address. For each IPv6 static route associated with the router with the nexthop IP: *G*, a priority−200 flow for IPv6 packets with match **eth.dst == 00:00:00:00:00:00 && xxreg0 ==** *G* with the following actions is added:

  ```
  nd_ns {
    eth.dst = E;
    ip6.dst = I
    nd.target = G;
    output;
  };
  ```

  Where *E* is the multicast mac derived from the Gateway IP, *I* is the solicited-node multicast address corresponding to the target address *G*.

  Unknown MAC address. A priority−100 flow for IPv6 packets with match **eth.dst == 00:00:00:00:00:00** has the following actions:

  ```
  nd_ns {
    nd.target = xxreg0;
    output;
  };
  ```

(Ingress table **IP Routing** initialized **reg1** with the IP address owned by **outport** and **(xx)reg0** with the next-hop IP address)

The IP packet that triggers the ARP/IPv6 NS request is dropped.

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

Egress Table 0: UNDNAT

This is for already established connections' reverse traffic. i.e., DNAT has already been done in ingress pipeline and now the packet has entered the egress pipeline as part of a reply. For NAT on a distributed router, it is unDNATted here. For Gateway routers, the unDNAT processing is carried out in the ingress DNAT table.

- For all the configured load balancing rules for a router with gateway port in **OVN_Northbound** database that includes an IPv4 address **VIP**, for every backend IPv4 address *B* defined for the **VIP** a priority−120 flow is programmed on **redirect−chassis** that matches **ip && ip4.src ==** *B* **&& outport ==** *GW*, where *GW* is the logical router gateway port with an action **ct_dnat;**. If the backend IPv4 address *B* is also configured with L4 port *PORT* of protocol *P*, then the match also includes **P.src** == *PORT*. These flows are not added for load balancers with IPv6 *VIPs*.

  If the router is configured to force SNAT any load-balanced packets, above action will be replaced by **flags.force_snat_for_lb = 1; ct_dnat;**.

- For each configuration in the OVN Northbound database that asks to change the destination IP address of a packet from an IP address of *A* to *B*, a priority−100 flow matches **ip && ip4.src ==** *B* **&& outport ==** *GW*, where *GW* is the logical router gateway port, with an action **ct_dnat;**.

  If the NAT rule cannot be handled in a distributed manner, then the priority−100 flow above is only programmed on the **redirect−chassis**.

  If the NAT rule can be handled in a distributed manner, then there is an additional action **eth.src =** *EA***;**, where *EA* is the ethernet address associated with the IP address *A* in the NAT rule. This allows upstream MAC learning to point to the correct chassis.

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

Egress Table 1: SNAT

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

Egress Table 1: SNAT on Gateway Routers

- If the Gateway router in the OVN Northbound database has been configured to force SNAT a packet (that has been previously DNATted) to *B*, a priority−100 flow matches **flags.force_snat_for_dnat == 1 && ip** with an action **ct_snat(***B***);**.

  If the Gateway router in the OVN Northbound database has been configured to force SNAT a packet (that has been previously load-balanced) to *B*, a priority−100 flow matches **flags.force_snat_for_lb == 1 && ip** with an action **ct_snat(***B***);**.

  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: SNAT on Distributed Routers

- 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* **&& outport ==** *GW*, where *GW* is the logical router gateway port, 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.

If the NAT rule cannot be handled in a distributed manner, then the flow above is only programmed on the **redirect−chassis**.

If the NAT rule can be handled in a distributed manner, then there is an additional action **eth.src =** *EA***;**, where *EA* is the ethernet address associated with the IP address *A* in the NAT rule. This allows upstream MAC learning to point to the correct chassis.

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

Egress Table 2: Egress Loopback

For distributed logical routers where one of the logical router ports specifies a **redirect−chassis**.

Earlier in the ingress pipeline, some east-west traffic was redirected to the **chassisredirect** port, based on flows in the **UNSNAT** and **DNAT** ingress tables setting the **REGBIT_NAT_REDIRECT** flag, which then triggered a match to a flow in the **Gateway Redirect** ingress table. The intention was not to actually send traffic out the distributed gateway port instance on the **redirect−chassis**. This traffic was sent to the distributed gateway port instance in order for DNAT and/or SNAT processing to be applied.

While UNDNAT and SNAT processing have already occurred by this point, this traffic needs to be forced through egress loopback on this distributed gateway port instance, in order for UNSNAT and DNAT processing to be applied, and also for IP routing and ARP resolution after all of the NAT processing, so that the packet can be forwarded to the destination.

This table has the following flows:

- For each NAT rule in the OVN Northbound database on a distributed router, a priority−100 logical flow with match **ip4.dst ==** *E* **&& outport ==** *GW*, where *E* is the external IP address specified in the NAT rule, and *GW* is the logical router distributed gateway port, with the following actions:

  ```
  clone {
     ct_clear;
     inport = outport;
     outport = "";
     flags = 0;
     flags.loopback = 1;
     reg0 = 0;
     reg1 = 0;
     ...
     reg9 = 0;
     REGBIT_EGRESS_LOOPBACK = 1;
     next(pipeline=ingress, table=0);
  };
  ```

  flags.loopback is set since in_port is unchanged and the packet may return back to that port after NAT processing. **REGBIT_EGRESS_LOOPBACK** is set to indicate that egress loopback has occurred, in order to skip the source IP address check against the router address.

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

Egress Table 3: 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;**.