NAME

ovn-trace – Open Virtual Network logical network tracing utility

SYNOPSIS

ovn−trace [*options*] *[datapath] microflow*

ovn−trace [*options*] **−−detach**

DESCRIPTION

This utility simulates packet forwarding within an OVN logical network. It can be used to run through "what-if" scenarios: if a packet originates at a logical port, what will happen to it and where will it ultimately end up? Users already familiar with the Open vSwitch **ofproto/trace** command described in **ovs−vswitch**(8) will find **ovn−trace** to be a similar tool for logical networks.

ovn−trace works by reading the **Logical_Flow** and other tables from the OVN southbound database (see **ovn−sb**(5)). It simulates a packet's path through logical networks by repeatedly looking it up in the logical flow table, following the entire tree of possibilities.

ovn−trace simulates only the OVN logical network. It does not simulate the physical elements on which the logical network is layered. This means that, for example, it is unimportant how VMs are distributed among hypervisors, or whether their hypervisors are functioning and reachable, so **ovn−trace** will yield the same results regardless. There is one important exception: **ovn−northd**, the daemon that generates the logical flows that **ovn−trace** simulates, treats logical ports differently based on whether they are up or down. Thus, if you see surprising results, ensure that the ports involved in a simulation are up.

The simplest way to use **ovn−trace** is to provide the *microflow* (and optional *datapath*) arguments on the command line. In this case, it simulates the behavior of a single packet and exits. For an alternate usage model, see **Daemon Mode** below.

The optional *datapath* argument specifies the name of a logical datapath. Acceptable names are the **name** from the northbound **Logical_Switch** or **Logical_Router** table, the UUID of a record from one of those tables, or the UUID of a record from the southbound **Datapath_Binding** table. (The **datapath** is optional because **ovn−trace** can figure it out from the *inport* that the *microflow* matches.)

The *microflow* argument describes the packet whose forwarding is to be simulated, in the syntax of an OVN logical expression, as described in **ovn−sb**(5), to express constraints. The parser understands prerequisites; for example, if the expression refers to **ip4.src**, there is no need to explicitly state **ip4** or **eth.type == 0x800**.

For reasonable L2 behavior, the microflow should include at least **inport** and **eth.dst**, plus **eth.src** if port security is enabled. For example:

inport == "lp11" && eth.src == 00:01:02:03:04:05 && eth.dst == ff:ff:ff:ff:ff:ff

For reasonable L3 behavior, *microflow* should also include **ip4.src** and **ip4.dst** (or **ip6.src** and **ip6.dst**) and **ip.ttl**. For example:

inport == "lp111" && eth.src == f0:00:00:00:01:11 && eth.dst == 00:00:00:00:ff:11
&& ip4.src == 192.168.11.1 && ip4.dst == 192.168.22.2 && ip.ttl == 64

Here's an ARP microflow example:

inport == "lp123"
&& eth.dst == ff:ff:ff:ff:ff:ff && eth.src == f0:00:00:00:01:11
&& arp.op == 1 && arp.sha == f0:00:00:00:01:11 && arp.spa == 192.168.1.11
&& arp.tha == ff:ff:ff:ff:ff:ff && arp.tpa == 192.168.2.22

ovn−trace will reject erroneous microflow expressions, which beyond syntax errors fall into two categories. First, they can be ambiguous. For example, **tcp.src == 80** is ambiguous because it does not state IPv4 or IPv6 as the Ethernet type. **ip4 && tcp.src > 1024** is also ambiguous because it does not constrain bits of **tcp.src** to particular values. Second, they can be contradictory, e.g. **ip4 && ip6**.

OUTPUT

ovn−trace supports the three different forms of output, each described in a separate section below. Regardless of the selected output format, **ovn−trace** starts the output with a line that shows the microflow being traced in OpenFlow syntax.

Detailed Output

The detailed form of output is also the default form. This form groups output into sections headed up by the ingress or egress pipeline being traversed. Each pipeline lists each table that was visited (by number and name), the **ovn−northd** source file and line number of the code that added the flow, the match expression and priority of the logical flow that was matched, and the actions that were executed.

The execution of OVN logical actions naturally forms a ‘‘control stack’’ that resembles that of a program in conventional programming languages such as C or Java. Because the **next** action that calls into another logical flow table for a lookup is a recursive construct, OVN ‘‘programs’’ in practice tend to form deep control stacks that, displayed in the obvious way using additional indentation for each level, quickly use up the horizontal space on all but the widest displays. To make detailed output more readable, without loss of generality, **ovn−trace** omits indentation for ‘‘tail recursion,’’ that is, when **next** is the last action in a logical flow, it does not indent details of the next table lookup more deeply. Output still uses indentation when it is needed for clarity.

OVN ‘‘programs’’ traces also tend to encounter long strings of logical flows with match expression **1** (which matches every packet) and the single action **next;**. These are uninteresting and merely clutter output, so **ovn−trace** omits them entirely even from detailed output.

The following excerpt from detailed **ovn−trace** output shows a section for a packet traversing the ingress pipeline of logical datapath **ls1** with ingress logical port **lp111**. The packet matches a logical flow in table 0 (aka **ls_in_port_sec_l2**) with priority 50 and executes **next(1);** to pass to table 1. Tables 1 through 11 are trivial and omitted. In table 19 (aka **ls_in_l2_lkup**), the packet matches a flow with priority 50 based on its Ethernet destination address and the flow’s actions output the packet to the **lrp11−attachement** logical port.

```
ingress(dp="ls1", inport="lp111")
---------------------------------
0. ls_in_port_sec_l2: inport == "lp111", priority 50
next(1);
19. ls_in_l2_lkup: eth.dst == 00:00:00:00:ff:11, priority 50
outport = "lrp11-attachment";
output;
```

Summary Output

Summary output includes the logical pipelines visited by a packet and the logical actions executed on it. Compared to the detailed output, however, it removes details of tables and logical flows traversed by a packet. It uses a format closer to that of a programming language and does not attempt to avoid indentation. The summary output equivalent to the above detailed output fragment is:

```
ingress(dp="ls1", inport="lp111") {
outport = "lrp11-attachment";
output;
...
};
```

Minimal Output

Minimal output includes only actions that modify packet data (not including OVN registers or metadata such as **outport**) and **output** actions that actually deliver a packet to a logical port (excluding patch ports). The operands of actions that modify packet data are displayed reduced to constants, e.g. **ip4.dst = reg0;** might be show as **ip4.dst = 192.168.0.1;** if that was the value actually loaded. This yields output even simpler than the summary format. (Users familiar with Open vSwitch may recognize this as similar in spirit to

the datapath actions listed at the bottom of **ofproto/trace** output.)

The minimal output format reflects the externally seen behavior of the logical networks more than it does the implementation. This makes this output format the most suitable for use in regression tests, because it is least likely to change when logical flow tables are rearranged without semantic change.

STATEFUL ACTIONS

Some OVN logical actions use or update state that is not available in the southbound database. **ovn−trace** handles these actions as described below:

ct_next
: By default **ovn−trace** treats flows as ‘‘tracked’’ and ‘‘established.’’ See the description of the **−−ct** option for a way to override this behavior.

ct_dnat (without an argument)
: Forks the pipeline. In one fork, advances to the next table as if **next;** were executed. The packet is not changed, on the assumption that no NAT state was available. In the other fork, the pipeline continues without change after the **ct_dnat** action.

ct_snat (without an argument)
: This action distinguishes between gateway routers and distributed routers. A gateway router is defined as a logical datapath that contains an **l3gateway** port; any other logical datapath is a distributed router. On a gateway router, **ct_snat;** is treated as a no-op. On a distributed router, it is treated the same way as **ct_dnat;**.

ct_dnat(*ip*)
ct_snat(*ip*)
: Forks the pipeline. In one fork, sets **ip4.dst** (or **ip4.src**) to *ip* and **ct.dnat** (or **ct.snat**) to 1 and advances to the next table as if **next;** were executed. In the other fork, the pipeline continues without change after the **ct_dnat** (or **ct_snat**) action.

ct_lb;
ct_lb(*ip*[**:***port*]...)**;**
: Forks the pipeline. In one fork, sets **ip4.dst** (or **ip6.dst**) to one of the load-balancer addresses and the destination port to its associated port, if any, and sets **ct.dnat** to 1. With one or more arguments, gives preference to the address specified on **−−lb−dst**, if any; without arguments, uses the address and port specified on **−−lb−dst**. In the other fork, the pipeline continues without change after the **ct_lb** action.

ct_commit
put_arp
put_nd These actions are treated as no-ops.

DAEMON MODE

If **ovn−trace** is invoked with the **−−detach** option (see **Daemon Options**, below), it runs in the background as a daemon and accepts commands from **ovs−appctl** (or another JSON-RPC client) indefinitely. The currently supported commands are described below.

trace [*options*] [*datapath*] *microflow*
: Traces *microflow* through *datapath* and replies with the results of the trace. Accepts the *options* described under **Trace Options** below.

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

OPTIONS

Trace Options

−−detailed
−−summary
−−minimal
: These options control the form and level of detail in **ovn−trace** output. If more than one of these options is specified, all of the selected forms are output, in the order listed above, each headed by a banner line. If none of these options is given, **−−detailed** is the default. See **Output**, above, for a

description of each kind of output.

−−all Selects all three forms of output.

−−ovs[**=***remote*]

Makes **ovn−trace** attempt to obtain and display the OpenFlow flows that correspond to each OVN logical flow. To do so, **ovn−trace** connects to *remote* (by default, **unix:/br−int.mgmt**) over OpenFlow and retrieves the flows. If *remote* is specified, it must be an active OpenFlow connection method described in **ovsdb**(7).

To make the best use of the output, it is important to understand the relationship between logical flows and OpenFlow flows. **ovn−architecture**(7), under **Architectural Physical Life Cycle of a Packet**, describes this relationship. Keep in mind the following points:

- **ovn−trace** currently shows all the OpenFlow flows to which a logical flow corresponds, even though an actual packet ordinarily matches only one of these.
- Some logical flows can map to the Open vSwitch "conjunctive match" extension (see **ovs−fields**(7)). Currently **ovn−trace** cannot display the flows with **conjunction** actions that effectively produce the **conj_id** match.
- Some logical flows may not be represented in the OpenFlow tables on a given hypervisor, if they could not be used on that hypervisor.
- Some OpenFlow flows do not correspond to logical flows, such as OpenFlow flows that map between physical and logical ports. These flows will never show up in a trace.
- When **ovn−trace** omits uninteresting logical flows from output, it does not look up the corresponding OpenFlow flows.

−−ct=*flags*

This option sets the **ct_state** flags that a **ct_next** logical action will report. The *flags* must be a comma- or space-separated list of the following connection tracking flags:

- **trk**: Include to indicate connection tracking has taken place. (This bit is set automatically even if not listed in *flags*.
- **new**: Include to indicate a new flow.
- **est**: Include to indicate an established flow.
- **rel**: Include to indicate a related flow.
- **rpl**: Include to indicate a reply flow.
- **inv**: Include to indicate a connection entry in a bad state.
- **dnat**: Include to indicate a packet whose destination IP address has been changed.
- **snat**: Include to indicate a packet whose source IP address has been changed.

The **ct_next** action is used to implement the OVN distributed firewall. For testing, useful flag combinations include:

- **trk,new**: A packet in a flow in either direction through a firewall that has not yet been committed (with **ct_commit**).
- **trk,est**: A packet in an established flow going out through a firewall.
- **trk,rpl**: A packet coming in through a firewall in reply to an established flow.
- **trk,inv**: An invalid packet in either direction.

A packet might pass through the connection tracker twice in one trip through OVN: once following egress from a VM as it passes outward through a firewall, and once preceding ingress to a second VM as it passes inward through a firewall. Use multiple **−−ct** options to specify the flags for multiple **ct_next** actions.

When **−−ct** is unspecified, or when there are fewer **−−ct** options than **ct_next** actions, the *flags* default to **trk,est**.

−−lb−dst=*ip*[**:***port*]
Sets the IP from VIP pool to use as destination of the packet. **−−lb−dst** is not available in daemon mode.

−−select−id=*id*
Specify the *id* to be selected by the **select** action. *id* must be one of the values listed in the **select** action. Otherwise, a random id is selected from the list, as if **−−select−id** were not specified. **−−select−id** is not available in daemon mode.

−−friendly−names
−−no−friendly−names
When cloud management systems such as OpenStack are layered on top of OVN, they often use long, human-unfriendly names for ports and datapaths, for example, ones that include entire UUIDs. They do usually include friendlier names, but the long, hard-to-read names are the ones that appear in matches and actions. By default, or with **−−friendly−names**, **ovn−trace** substitutes these friendlier names for the long names in its output. Use **−−no−friendly−names** to disable this behavior; this option might be useful, for example, if a program is going to parse **ovn−trace** output.

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 .

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 determined 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 **/usr/local/var/log/ovn/***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 to use SSL for the connection to the database (and the switch, if **−−ovs** is specified).

−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

−−db *database*
The OVSDB database remote to contact. If the **OVN_SB_DB** environment variable is set, its value is used as the default. Otherwise, the default is **unix:/db.sock**, but this default is unlikely to be useful outside of single-machine OVN test environments.

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

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