Protocol validation

in OVS + OVN test suites

Ihar Hrachyshka

Brief

Define the topic.

Why it's important.

Some useful tools.

Protocol validation

as in

"hopefully, behavior reflects RFC"

How we test a new feature?

Ihar Hrachyshka

How we test a new feature?

- we can create object X **in NB**...

How we test a new feature?

- we can create object X **in NB**...
- that is translated into object Y **in SB**...

How we test a new feature?

- we can create object X **in NB**...
- that is translated into object Y **in SB**...
- that is translated into **OpenFlow** flows...

How we test a new feature?

- we can create object X **in NB**…
- that is translated into object Y **in SB**…
- that is translated into **OpenFlow** flows…
- **We often stop here.**

Deeper?

We can check the path with *ovn-trace* or `ovs-ofctl ofproto/trace`.

But it reflects **intent**, not **reality**. The packet is not injected.

And how to validate a reply to injected packet?

Why would we care?

Why would we care?

- End-to-end proof it **actually** works

Why would we care?

- End-to-end proof it **actually** works
- These bugs are **hard** to debug in field

Why would we care?

- End-to-end proof it **actually** works
- These bugs are **hard** to debug in field
    - pcaps

Why would we care?

- End-to-end proof it **actually** works
- These bugs are **hard** to debug in field
  - pcaps
  - wireshark

Why would we care?

- End-to-end proof it **actually** works
- These bugs are **hard** to debug in field
    - pcaps
    - wireshark
    - animal bones

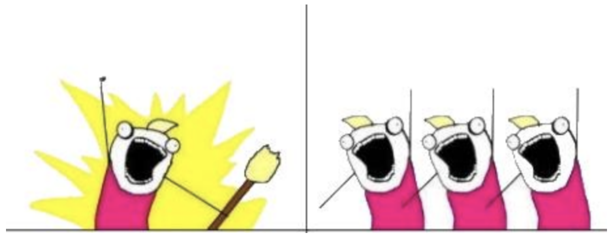
Why would we care?

- End-to-end proof it **actually** works
- These bugs are **hard** to debug in field
    - pcaps
    - wireshark
    - animal bones
    - voodoo dolls

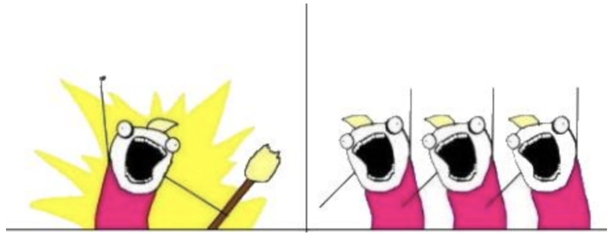
Why would we care?

- End-to-end proof it **actually** works
- These bugs are **hard** to debug in field
  - pcaps
  - wireshark
  - animal bones
  - voodoo dolls
- **Technically correct** is the best kind of **correct**

What do we want?

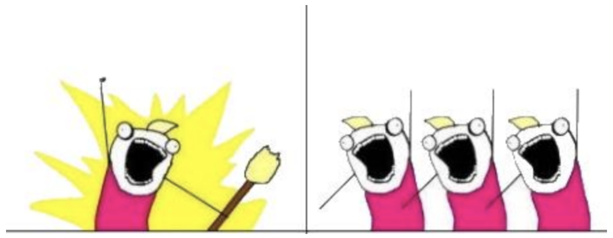


What do we want?



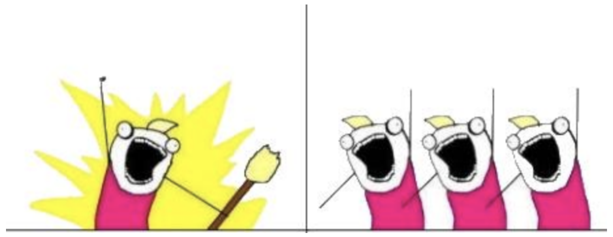
- **prepare** a packet

What do we want?



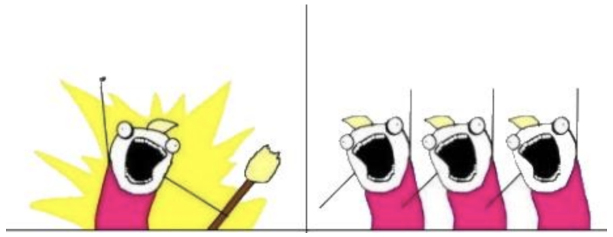
- **prepare** a packet
- **inject** it into dataplane

What do we want?



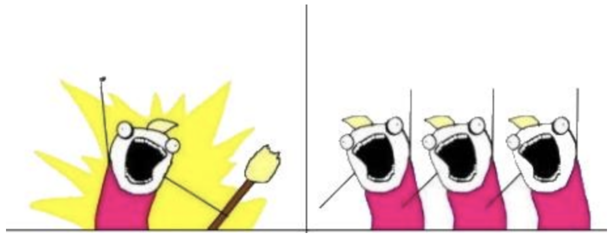
- **prepare** a packet
- **inject** it into dataplane
- **receive** it on the other end

What do we want?



- **prepare** a packet
- **inject** it into dataplane
- **receive** it on the other end
    - also a peer **reply**?

What do we want?



- **prepare** a packet
- **inject** it into dataplane
- **receive** it on the other end
  - also a peer **reply**?
- confirm observed **matches** expected

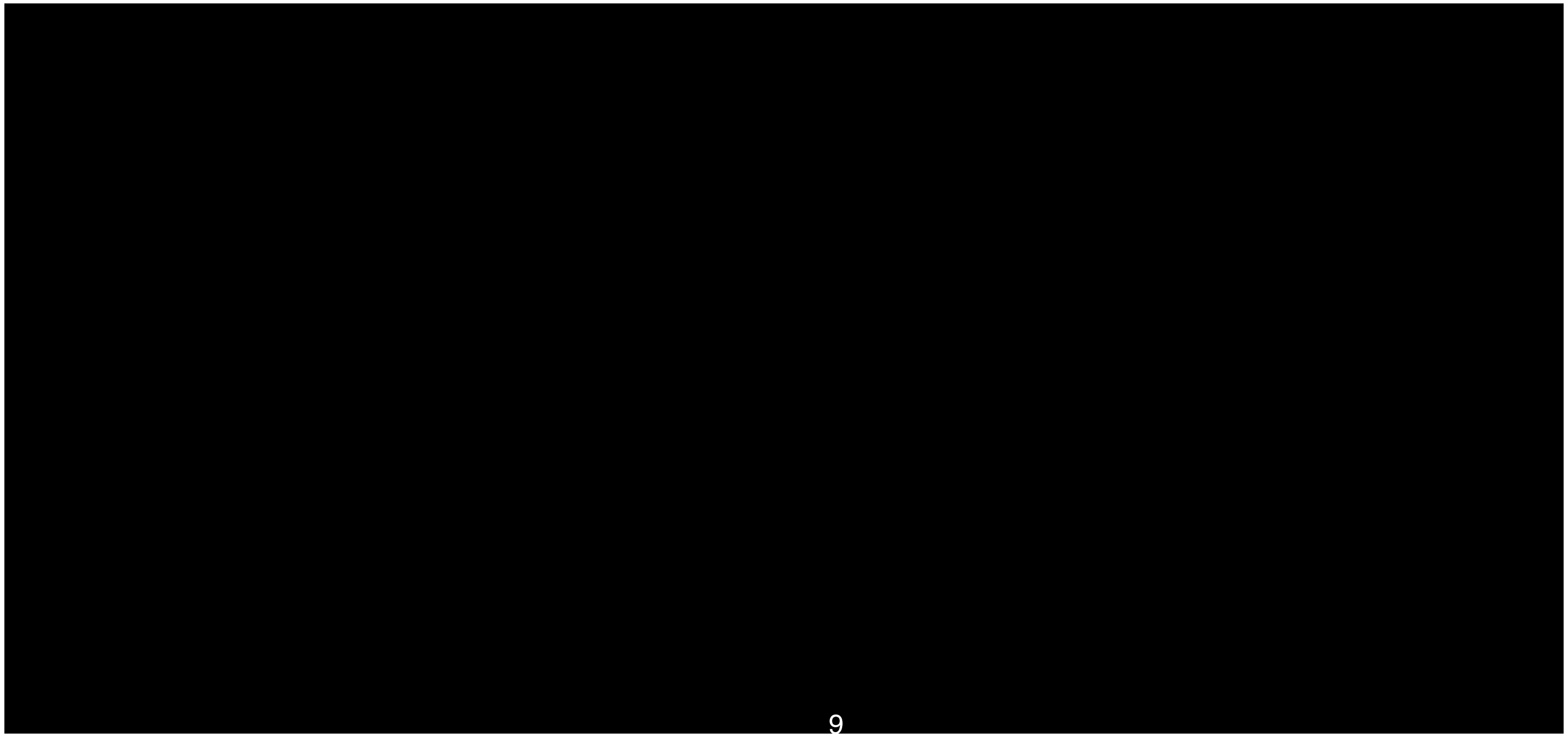
ovs: netdev-dummy/receive

```
AT_CHECK([ovs-appctl netdev-dummy/receive port ${packet}])
```

${packet}

```
local packet="...0800450000280000 4000ff06000000001000100000..."

AT_CHECK([ovs-appctl netdev-dummy/receive port ${packet}])
```



Ihar Hrachyshka

ovn-controller: inject-pkt

```
packet='inport=="$inport" &&
        eth.src==8a:bf:7e:2f:05:84 &&
        eth.dst==0a:8f:39:4f:e0:73 &&
        ip4 && ip.ttl==64 &&
        ip4.src==192.168.123.2 &&
        ip4.dst==192.168.123.1 &&
        udp && udp.src==53 && udp.dst==4369'

OVS_WAIT_UNTIL([as hv ovs-appctl -t ovn-controller \
                inject-pkt "$packet"])
```

inject-pkt - caveats

- `ovn-controller` only
- one packet per main loop iteration
- limited to OVN logical flow syntax

ovs: ovs-ofctl compose-packet

```
flow="\
  eth_src=8a:bf:7e:2f:05:84,\
  eth_dst=0a:8f:39:4f:e0:73,\
  dl_type=0x0800,\
  nw_src=192.168.123.2,\
  nw_dst=192.168.123.1,\
  nw_proto=6,nw_ttl=64,nw_frag=no,\
  tp_src=54392,tp_dst=5201,tcp_flags=ack"

packet=`ovs-ofctl compose-packet --bare "${flow}"`
```

Ihar Hrachyshka

ovs: options:tx_pcap=

```
# capture pcap
AT_CHECK([ovs-vsctl set Interface port2 options:tx_pcap=out.pcap])

# inject packet
packet="..."
AT_CHECK([ovs-appctl netdev-dummy/receive port1 ${packet}])

# confirm received
AT_CHECK([ovs-pcap out.pcap > out.pcap.txt 2>&1])
AT_CHECK_UNQUOTED([tail -n 1 out.pcap.txt], [0], [${packet}
])
```

compose-packet: NAT

```
# pre-NAT
flow="..."
packet=`ovs-ofctl compose-packet --bare "${flow}"`

# post-NAT
expected_flow=`echo "${flow}" | sed 's/192.168.1.1/8.8.8.1/g'`
expected=`ovs-ofctl compose-packet --bare "${expected_flow}"`

AT_CHECK([ovs-appctl netdev-dummy/receive port1 ${packet}])

AT_CHECK([ovs-pcap port2.pcap > port2.pcap.txt 2>&1])
AT_CHECK_UNQUOTED([tail -n 1 port2.pcap.txt], [0], [${expected}
])
```

compose-packet: bad checksum

```
flow="..."

packet=`ovs-ofctl compose-packet --bare --bad-csum "${flow}"`
```

fmt_pkt

Ihar Hrachyshka

fmt_pkt

- when `compose-packet` is not enough (**L4+**)

`fmt_pkt`

- when `compose-packet` is not enough (**L4+**)
- available as part of `ovn-macros.at`

`fmt_pkt`

- when `compose-packet` is not enough (**L4+**)
- available as part of `ovn-macros.at`
- based on `scapy`

fmt_pkt: DHCPv6 example

```
local packet="Ether(dst='ff:ff:ff:ff:ff:ff', src='${src_mac}')/
                    IPv6(dst='ff02::1:2', src='${src_lla}')/
                    UDP(sport=546, dport=547)/
                    DHCP6(msgtype=${msg_code}, trid=0x010203)/
                    DHCP6OptClientId(
                        duid=DUID_LL(lladdr='${src_mac}'))"

as hv1 ovs-appctl netdev-dummy/receive port `fmt_pkt $packet`
```

Docs at https://scapy.readthedocs.io…

…or just play in REPL

```
$ python3
>>> from scapy.all import *

>>> [x for x in dir() if "ARP" in x]
['ARP', 'ARPHDR_ETHER', 'ARPHDR_LOOPBACK', ...]

>>> help(ARP)
...
class ARP(scapy.packet.Packet)
 |   ARP(_pkt, /, *, hwtype=1, ptype=2048, hwlen=None, plen=None,
         op=1, hwsrc=None, psrc=None, hwdst=None, pdst=None)
...
```

fmt_pkt v0.0.1

```
fmt_pkt() {
    echo "from scapy.all import *; \
          import binascii; \
          out = binascii.hexlify(raw($1)); \
          print(out.decode())" | $PYTHON3
}
```

Ihar Hrachyshka

fmt_pkt v0.0.2

```
fmt_pkt() {
    ctlfile=$ovs_base/scapy.ctl
    if  ! -e $ctlfile ; then
        start_scapy_server
    fi
    ovs-appctl -t $ctlfile \
        payload "$1"
}
start_scapy_server() {
    ctlfile=$ovs_base/scapy.ctl
    "$top_srcdir"/tests/scapy-server.py \
        --unixctl=$ctlfile \
        --log-file=$ovs_base/scapy.log ...
    on_exit "... && ovs-appctl -t $ctlfile exit"
}
```

fmt_pkt gotchas

Ihar Hrachyshka

fmt_pkt gotchas

- max 10 requests for python `unixctl` AF_UNIX servers

fmt_pkt gotchas

- max 10 requests for python `unixctl` AF_UNIX servers
  - don't run in background - & (yet)

fmt_pkt gotchas

- max 10 requests for python `unixctl` AF_UNIX servers
  - don't run in background - `&` (yet)
- `scapy` is powerful, but not almighty

fmt_pkt gotchas

- max 10 requests for python `unixctl` AF_UNIX servers
    - don't run in background - `&` (yet)
- `scapy` is powerful, but not almighty
    - but `raw()` accepts any `python` code, e.g. `socket.in6_getnsma`

fmt_pkt gotchas

- max 10 requests for python `unixctl` AF_UNIX servers
  - don't run in background - `&` (yet)
- `scapy` is powerful, but not almighty
  - but `raw()` accepts any `python` code, e.g. `socket.in6_getnsma`
- still slower than `ovs-ofctl compose-packet`

compose-packet or fmt_pkt?

L3? `compose-packet`

L4+? `fmt_pkt` (maybe)

feedback and questions welcome

Ihar Hrachyshka