Troubleshooting demonstration tutorial from the AutoTPG side



Figure 1: Troubleshooting Demonstration

The code for the troubleshooting demo is available in the kurgan machine (kurgan.testbed.se, username:unify and password:unify-demo-stress-out).

The docker image for installing the autoTPG controller is also available at:

<https://github.com/stevenvanrossem/un-orchestrator/tree/elastic-router/use-cases/elastic_router/autoTPG>

The following actions are performed for running the troubleshooting demonstration:

1. Troubleshooting start up actions
2. RQL start up actions
3. AutoTPG start up actions

### Troubleshooting start up Action:

Troubleshooting starts, when EPOXIDE receives a troubleshooting request from the aggregator DD client (Step 3 in Figure 1). At this time, EPOXODE queries the local orchestrator for the NFFG file (Step 4a). The format of the NFFG file is given by:

{

"forwarding-graph": {

"id": "00000001",

"name": "Elastic Router graph",

"VNFs": [

{

"id": "00000001",

"name": "ctrl",

"unify-env-variables": [

{

"variable": "CFOR=172.17.0.1:8080"

}

],

"unify-control": [

{

"host-tcp-port": 5000,

"vnf-tcp-port": 5000

}

],

"ports": [

{

"id": "inout:0",

"name": "control-port",

"unify-ip": "10.0.10.100/24"

}

]

},

{

"id": "00000002",

"name": "ovs1",

"unify-env-variables": [

{

"variable": "CONTROLLER=tcp:10.0.10.100:6633"

},

{

"variable": "VNF\_NAME=ovs1"

},

{

"variable": "OVS\_DPID=9900000000000001"

}

],

"unify-control": [

{

"host-tcp-port": 9001,

"vnf-tcp-port": 22

}

],

…….

From the above NFFG file, EPOXIDE knows the following information:

1. host-tcp-port information to access all vswitchd VNFs (e.g., 9001 for ovs1) through the public IP address of the UN node.
2. Controller information (e.g., tcp:10.0.10.100:6633)
3. The vswitchd datapath IDs (e.g., 9900000000000001)

### RQL Start up Actions

In Step 5 (Figure 1), EPOXIDE invokes RQL and queries DBs for finding the switches where verification through AutoTPG is required (Step 6).

### AutoTPG Start up actions

In Step 7 (Figure 1), EPOXIDE invokes the autoTPG tool with the information received in Step 4a, emulates errors, and runs commands to verify a switch. The following commands can be used to verify ovs1 with the information available through the NFFG file given above.

1. SSH PUBLIC\_IP\_ADDRESS\_UN -p 9001
2. Run the following command to configure the autoTPG controller in ovs1 from the kurgan machine (or developer laptop)

ovs-vsctl set-controller ovs1 tcp:10.0.10.100:6633 tcp:OWN \_PUBLIC\_ADDRESS:7777

1. Run the autoTPG controller by following commands:
2. cd /home/unify/OpenSource-AutoTPG/floodlight-plus/
3. sudo ./floodlight.sh
4. Wait until the ovs1 vswitchd is discovered by the controller by reading “Switch (dp\_id=99:00:00:00:00:00:00:01) added” in the logfile of the controller
5. Emulate a matching error by running the following command:

curl -d '{"switch":"99:00:00:00:00:00:00:01", "tableid": "0", "priority":"1001","manual":"true", "etherType":"0x800","ipSrc":"11.0.0.2/32", ipDst":"10.0.1.2/32", "IPProto":"0", "srcport":"1", "dstport":"2", "output":"2", "add":"true"}' <http://127.0.0.1:8080/wm/autoTPG/qpc/json>

1. Here, the datapath id of the switch is {"switch":"99:00:00:00:00:00:00:01",.
2. ID of the table is 0 (i.e., "tableid": "0")
3. Priority is 1001 ("priority":"1001")
4. The flow is considered manual because (manual:true)
5. The EtherType is 0x800 ("etherType":"0x800")
6. The IP address of the source is 11.0.0.2/32 (ipSrc":"11.0.0.2/32")
7. The IP address of the destination is “10.0.1.2/32” (ipDst":"10.0.1.2/32")
8. The protocol number is 0 ("IPProto":"0")
9. The transport layer source port is 1 ("srcport":"1")
10. The transport layer destination port is 2 ( "dstport":"2")
11. The outgoing port is 2 ("output":"2")
12. This flow should be added in the switch (as “add:”true”) or this flow should not be added in the switch (“add:”false”). However, when add is false, administrator should manually add the Flow Entry (described by the above command).

If any of the field of the matching header is not defined in the above command, the field is considered wildcarded. The example of such a command is:

curl -d '{"switch":"99:00:00:00:00:00:00:01", "tableid": "0", "priority":"1001","manual":"true", "etherType":"0x800", "ipDst":"10.0.1.2/32", "output":"2", "add":"true"}' <http://127.0.0.1:8080/wm/autoTPG/qpc/json>.

In this example, IP source, protocol number, source port and dstport is wildcarded.

1. After the above command, the following command can be run to verify a vswitchd VNF ( in this case vSwitchd VNF is ovs1)

curl -d '{"switch":"99:00:00:00:00:00:00:01", "method":"bin"}' <http://127.0.0.1:8080/wm/autoTPG/qpvf/json>

1. The results are then shown in the same terminal. The format of the results are given by:

{"errors":1,"exitcode":"SUCCESS","culprit":["10.0.1.2"],"logfile":"/tmp/error.txt"}

The file /tmp/error.txt can be viewed to see a detail about errors.

### Description about emulating errors in the integration demo:

Currently, we are emulating matching errors by: (1) adding manually a Flow Entry with a high priority and (2) sending a REST command to the autoTPG controller that a particular Flow Entry entry (source IP, destination IP, protocol number, source port, and destination port) is not added the master controller. This REST command can be executed in the format given by paragraph E (above). When the above command is received the autoTPG controller, it does not include this particular Flow Entry to find that which Flow Entry should be matched with which packet.

For example, let us suppose there are the following two Flow Entry in a switch:

Priority EtherType Destination IP ------- output port

100 0x800 10.1.1.0 -------- 3

100 0x800 10.1.2.0 -------- 3

In order to emulate matching errors for a packet with the destination IP address 10.1.1.2 , the administration should manually add the following high priority Flow Entry:

1001 0x800 10.1.1.2 -------- 2

In addition, the administrator should send a rest API command to the autoTPG controller to inform it about the manual entry by:

curl -d '{"switch":"99:00:00:00:00:00:00:01", "tableid": "0", "priority":"1001","manual":"true", "etherType":"0x800", "ipDst":"10.0.1.2/32", "output":"2", "add":"true"}' <http://127.0.0.1:8080/wm/autoTPG/qpc/json>

After this, EPOXIDE can verify switch 99:00:00:00:00:00:00:01 for matching errors.