What is SONiC-P4 Software Switch

SONiC-P4 is a software switch that runs on the P4-emulated SAI behavioral model software switch ASIC (can be viewed here). It uses the sai_bm.p4 to program the P4-emulated switch ASIC to emulate the data plane behavior. On top of that, it runs the real SONiC network stack. The current SONiC-P4 is released as a docker image. You can run it anywhere that docker runs -- inside a bare-metal Linux/Windows machine, inside your favorite virtual machine, or inside your favorite cloud environment. However, because of this choice, we build the various SONiC modules in a single docker image, instead of having them as their own docker image. We find this a very useful tool for developing and testing upper layer features.

How to use SONiC-P4 Software Switch

In the following, we explain the usage of SONiC-P4 software switch with a very simple testbed.

Topology:

host1 (Ubuntu, 192.168.1.2/24) <--> switch1 (SONiC) <--> switch2 (SONiC) <--> host2 (Ubuntu, 192.168.2.2/24)

switch1 and switch2 are two SONiC-P4 switches in two different BGP AS, peering with each other. Switch1 announces 192.168.1.0/24, switch2 announces 192.168.2.0/24

On a Ubuntu server, download necessary files Here. Unzip the file and go to the sonic/ directory.

Run ./install_docker_ovs.sh to install docker and open-vswitch.

Run ./load_image.sh to load the SONiC-P4 image. Currently, we specify loading build 543. You may specify other builds later than 543 if needed.

Run ./start.sh to setup the testbed. You should see the four dockers when finish.

```
lgh@acs-p4:~/sonic$ docker ps
CONTAINER ID   IMAGE                    COMMAND       CREATED          STATUS          PORTS     NAMES
db924306352f   ubuntu:14.04             "/bin/bash"   2 minutes ago    Up 2 minutes              host2
ae5e987dee8c   ubuntu:14.04             "/bin/bash"   2 minutes ago    Up 2 minutes              host1
aed19d76cd3a   docker-sonic-p4:latest   "/bin/bash"   2 minutes ago    Up 2 minutes              switch2
680f95a83512   docker-sonic-p4:latest   "/bin/bash"   2 minutes ago    Up 2 minutes              switch1
Wait for ~60 seconds of bootup time. Then run ./test.sh, which pings host2 from host1.
lgh@acs-p4:~/sonic$ ./test.sh
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=62 time=9.81 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=62 time=14.9 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=62 time=8.42 ms
64 bytes from 192.168.2.2: icmp_seq=4 ttl=62 time=14.7 ms
Check BGP on switch1
lgh@acs-p4:~/sonic$ docker exec -it switch1 bash
root@switch1:/# vtysh -c "show ip bgp sum"
BGP router identifier 192.168.1.1, local AS number 10001
RIB entries 3, using 336 bytes of memory
Peers 1, using 4568 bytes of memory
```

Neighbor V AS MsgRcvd MsgSent TblVer InQ OutQ Up/Down State/PfxRcd

10.0.0.1 4 10002 1993 1996 0 0 0 00:33:12 1

Total number of neighbors 1

run ./stop.sh to cleanup.

Explanation of start.sh

We set up the topology in start.sh. First, we start four docker containers in the topology. Take the command for switch1 as an example.

sudo docker run --net=none --privileged --entrypoint /bin/bash --name switch1 -it -d -v $PWD/switch1:/sonic docker-sonic-p4:latest

We specify --net=none to prevent Docker engine from adding its docker0 interface, which may interfere with the topology being tested. --privileged is to enable each container to configure their own interfaces. -v $PWD/switch1:/sonic mounts the configuration folder into the switch containers.

Then we create three links. Take the link between switch1 and switch2 as an example. The following commands connect switch1's interface eth1 with switch2's interface eth1.

sudo ovs-vsctl add-br switch1_switch2

sudo ovs-docker add-port switch1_switch2 eth1 switch1

sudo ovs-docker add-port switch1_switch2 eth1 switch2

We also configure the interface IP and default routes on the host1 and host2. Take host1 as an example.

sudo docker exec -d host1 ifconfig eth1 192.168.1.2/24 mtu 1400

sudo docker exec -d host1 ip route replace default via 192.168.1.1

Finally, we invoke the startup script for switch1 and switch2.

sudo docker exec -d switch1 sh /sonic/scripts/startup.sh

sudo docker exec -d switch2 sh /sonic/scripts/startup.sh

SONiC-P4 Configuration Details

In start.sh, we have mounted the configuration folder into the switch container, at /sonic. The most important configurations are at /sonic/scripts/startup.sh, /sonic/etc/config_db/vlan_config.json, and /sonic/etc/quagga/bgpd.conf.

In /sonic/scripts/startup.sh, we start all SONiC services and a P4 software switch. The P4 software switch is started by this line (see supervisord.conf)

simple_switch --log-console -i 1@eth1 -i 2@eth2 ...

It binds interface eth1 to P4 software switch's port 1, eth2 to port 2, and so on. These ethX interfaces are usually referred as front-panel interfaces, and directly used by the P4 switches for carrying data plane packets. However, SONiC operates on another type of interfaces, called host interfaces. Host interfaces are for SONiC control plane, and are NOT carrying data plane packets. The host interfaces are named as EthernetX. We configure peer-to-peer IP and MTU on host interfaces. SONiC reads the configurations, like IP and MTU, from host interfaces, then configures these values on the P4 software switch using SAI. The mapping between host interfaces and switch ports is specified in /port_config.ini:

alias lanes

Ethernet0 1

Ethernet1 2

...

Together with the simple_switch command in /sonic/scripts/startup.sh, we have configured this mapping Ethernet0 --> lane 1 --> eth1. It is essentially a mapping between host interfaces and front-panel interfaces.

/sonic/etc/config_db/vlan_config.json configures the switch vlan interfaces used in this test, using ConfigDB interface, look here for details:


```
{
   "VLAN": {
      "Vlan15": {
         "members": [
            "Ethernet0"
         ],
         "vlanid": "15"
      },
      "Vlan10": {
         "members": [
            "Ethernet1"
         ],
         "vlanid": "10"
      }
   },
   "VLAN_MEMBER": {
      "Vlan15|Ethernet0": {
         "tagging_mode": "untagged"
      },
      "Vlan10|Ethernet1": {
         "tagging_mode": "untagged"
      }
   },
```

```
  "VLAN_INTERFACE": {
    "Vlan15|10.0.0.0/31": {},
    "Vlan10|192.168.1.1/24": {}
  }
}
```

/sonic/etc/quagga/bgpd.conf configures the BGP session on the switch. Here is the BGP configuration for switch1, which peers with switch2 using peer-to-peer IP 10.0.0.0/31, and announces 192.168.1.0/24.

```
router bgp 10001
 bgp router-id 192.168.1.1
 network 192.168.1.0 mask 255.255.255.0
 neighbor 10.0.0.1 remote-as 10002
 neighbor 10.0.0.1 timers 1 3
 neighbor 10.0.0.1 send-community
 neighbor 10.0.0.1 allowas-in
 maximum-paths 64
!
access-list all permit any
```