$ sudo mn

The default topology is the minimal topology, which includes one OpenFlow kernel switch

connected to two hosts, plus the OpenFlow reference controller.This topology could also be specified on the command line with --topo=minimal.Other topologies are also available out of the box; see the --topo section in the output of mn -h.

All four entities (2 host processes, 1 switch process, 1 basic controller) are now

running in the VM. The controller can be outside the VM, and instructions for that

are at the bottom.If no specific test is passed as a parameter, the Mininet CLI comes up.

In the Wireshark window, you should see the kernel switch connect to the reference controller.

mininet> h1 ifconfig -a

You should see the host’s h1-eth0 and loopback (lo) interfaces. Note that this interface (h1-eth0) is not seen by the primary Linux system when ifconfig is run, because it is specific to the network namespace of the host process.

In contrast, the switch by default runs in the root network namespace, so running a command on the “switch” is the same as running it from a regular terminal:

mininet> s1 ifconfig -a

This will show the switch interfaces, plus the VM’s connection out (eth0).

For other examples highlighting that the hosts have isolated network state, run arp and route on both s1 and h1.

It would be possible to place every host, switch and controller in

its own isolated network namespace, but there’s no real advantage to doing so,

unless you want to replicate a complex multiple-controller network.

Mininet does support this; see the --innamespace option.

Cleanup

If Mininet crashes for some reason, clean it up:

$ sudo mn –c

XTerm Display

For more complex debugging, you can start Mininet so that it spawns one or more xterms.

To start an xterm for every host and switch, pass the -x option:

$ sudo mn -x

After a second, the xterms will pop up, with automatically set window names.

Alternately, you can bring up additional xterms as shown below.

By default, only the hosts are put in a separate namespace; the window for each

switch is unnecessary (that is, equivalent to a regular terminal), but can be a

convenient place to run and leave up switch debug commands, such as flow counter

dumps.

Xterms are also useful for running interactive commands that you may need to cancel,

for which you’d like to see the output.

1. >dump

It shows network details

2.> pingall

3. h1 ping h2 or h1 ping 10.0.0.2

4. on x terminal - only use ping 10.0.0.2 on h1 terminal

5. On putty window where X11 is enabled and Xming server is on

To start wireshark- $sudo mn -x

6.$sudo mn

It creates a n/w of 2 hosts, 1 controller and 1 switch

These two switches are controlled by default controller of pyretic.

7. $sudo mn -x

It creates a n/w of 2 hosts, 1 controller and 1 switch

and opens these 4 x terminals

8. On simple putty window-

Write controller program in pyretic/pyretic folder

Start the controller by running that program

cmd: pyretic.py pyretic.program\_name

OR

pyretic.py -m program\_name.py

where -m=>mode

mode types-1) i - interpreted

2) r0 - reactive

3) p0 - proactive

e.g. pyretic.py -m i program\_name.py

9. mac\_learner() module witten by pyretic manages working of switch

Code for it is can be viewed as-

cmd: vi pyretic/pyretic/modules/mac\_learner.py

10.Logical separation of data plane and control plane is carried out when we write our own controller on the same machine.

cmd: $sudo mn --topo single,10 --mac -x --controller=remote

where,

--topo single,10 => This creates n/w of 10 hosts and 1 switch

--mac => Assigns the mac addresses serially to each host

-x => opens x terminals for all parts in n/w

--controller=remote => connects the n/w to the controller exist on localhost

Physical separation

$sudo mn --topo single,10 --mac -x --controller=remote,ip=127.0.0.1, port=6633