Mininet on OpenBSD

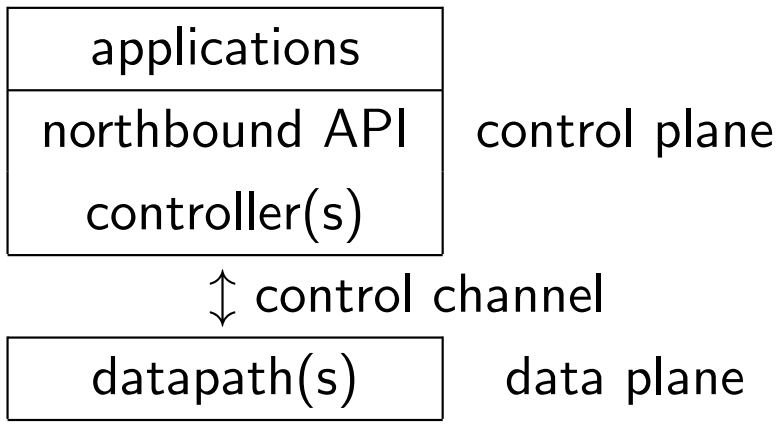
Using rdomains for Interactive SDN Testing and Development

Ayaka Koshibe
akoshibe@openbsd.org

AsiaBSDCon 2018

"SDN"?

- Network split into programmable nodes that handle traffic and entities that program them



OpenFlow

A control channel protocol standardized by the ONF

- datapath follows flow rules installed on one or more flow tables
  - flow/match: traffic class defined by packet header pattern
  - action: output to port/group, rewrite field, search another table...
- controller discovers datapath features from initial handshake, state from requests

OpenBSD and SDN

OpenBSD has its own OpenFlow 1.3 SDN stack since 6.1

- switch(4): datapath
  - switchN has /dev/switchN as its control channel
- switchd(8): controller
  - implements flow forwarding logic
  - can forward control messages to other controllers
- switchctl(8): control application for switchd(8)

Scenario

You are an SDN developer. How do you test your work?

- hardware testbeds?
- personal dev environment?

Mininet

An 'Emulator for rapid prototyping of Software Defined Networks'

- `mn` command to launch networks and run tests
- a set of APIs for scripting topologies and test scenarios
- CLI for topologies
- topology creation GUI (MiniEdit)

Basic Usage: mn command

Quick testing with built-in tests (ping, iperf)

- ping among hosts across a chain of three switches:

```
# mn --topo=linear,3 --test=pingall
*** Creating network
*** Adding controller

(...startup output)

*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)

(...teardown output)

completed in 0.383 seconds
```

Basic Usage: CLI

Launch a CLI to manipulate topology

- break links, run commands in nodes...

```
# mn --topo=linear,3 --verbosity=output
mininet> link s1 s2 down
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X h3
h3 -> X h2
*** Results: 66% dropped (2/6 received)
mininet> link s1 s2 up
mininet>
mininet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=3.97 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 3.976/3.976/3.976/0.000 ms
mininet>
```

Basic Usage: Python API

Create a custom topology:

```
$ cat test.py
#!/usr/bin/env python
# example using "high-level" API
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI

class MinimalTopo(Topo):
    def build(self):
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        s1 = self.addSwitch('s1')

        self.addLink(h1, s1)
        self.addLink(h2, s1)

net = Mininet(topo=MinimalTopo())
net.start()
CLI(net)
net.stop()
```

```
# ./test.py
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
mininet>
```

Basic Usage: Python API

Run commands for experiments:

- `cmd()`: run commands on a node
- `quietRun()`: run commands against the network

```
# build network of two hosts: h1--h2 ("mid-level" API example)
net = Mininet()
h1 = net.addHost('h1')
h2 = net.addHost('h2')
net.addLink(h1, h2)
net.start()

# start simple server in h2 and fetch page from h1
h2.cmd('python -m SimpleHTTPServer 80 &')
sleep(2)
print(h1.cmd('curl', h2.IP()))

# print interfaces on the host and exit
print(quietRun('ip link'))
net.stop()
```

Development Workflow

I have a...

controller/application:

- use a topology pointed at a running instance
  - `mn --controller=remote,ip=x.x.x.x,port=y`
  - `net.addController(controller=RemoteController)`
- add a custom controller node (`--controller=myctl`)

switch:

- add a custom vswitch node (`--switch=myswitch`)
- use a topology with a physical port wired to a switch

Internals: Mininet objects

- Mininet : coordinates the emulation process
- Topo : graph of nodes, ports(intfs), and links
  - Node : `bash` running interactively in network namespace
  - Intf : virtual ethernet (`veth`) interfaces
  - Link : pairs of Intfs created/configured with `iproute2`
- Switch : nodes running vswitches
  - OpenvSwitch(default), ofsoftswitch13, Linux bridge...
- Controller : nodes running controller applications
  - Stanford reference controller(default), Ryu, Nox...

Internals: Topology creation

```
*** Creating network
*** Adding controller
*** Adding hosts:
*** Adding switches:
mnexec bash --norc -is 'mininet:c0'
(repeat for h1,h2,s1)

*** Adding links:
ip link add name s1-eth1 type veth peer name h1-eth0
ip link set s1-eth1 netns <s1>
ip link set h1-eth0 netns <h1>
ifconfig s1-eth1 up
ifconfig h1-eth0 up
(repeat for s1-eth2 <-> h2-eth0)

*** Configuring hosts
ifconfig h1-eth0 10.0.0.1/8 up
(repeat for h2-eth0 at 10.0.0.2)

*** Starting controller
(in c0) controller -v ptcp:6653 1>/tmp/c0.log 2>/tmp/c0.log &

*** Starting 1 switches
(in s1) ovs-vsctl create Controller target="tcp:127.0.0.1:6653" ...

*** Starting CLI:
mininet>
```

Initial goals

- recreate core features ("base" Mininet)
  - topology emulation, CLI, remote controller
  - switchd(8) and switch(4) incorporated as nodes
- aim to eventually get it upstreamed
  - preserve Linux support (for github fork)

Minimum requirements

- network virtualization (separate address space), L2 and up
- vswitches and controllers for nodes
- applications for baseline tests

rdomain(4) and pair(4)

- a routing domain
  - provides separate network address spaces
  - recieves traffic via interfaces attached to them
  - can restrict a process and descendants to its address space
- a pair(4) interface
  - pairs with another to form endpoints of a virtual Ethernet link
  - can be attached to an rdomain

Implementation: Mininet objects

- Node: ksh running in a routing domain
- Switch: node dedicated to a switch(4) instance
  - switchd in forwarding mode for RemoteController case
- Controller: node running switchd(8)
  - uses Mininet-specific switchd.conf(5)
- Link: two patched pair(4)s

Implementation: A comparison

	Linux	OpenBSD
Hosts	bash setns(mnexec)	ksh route
Links	veth iproute2(ip link)	pair ifconfig
Switches	OVS ovs-vsctl/ovs-ofctl	switch switchctl, ifconfig
Controllers	controller	switchd + switchctl
Bridges	Linux bridge brctl	bridge ifconfig

Topology creation revisited

```
*** Creating network
*** Adding controller
*** Adding hosts:
*** Adding switches:
route -T <rdomain> exec /bin/ksh -is 'mininet:c0'
(repeat for h1,h2,s1)

*** Adding links:
ifconfig pair1 create rdomain <s1> up
ifconfig pair2 create rdomain <h1> patch pair1 up
ifconfig pair1 description 's1-eth1'
ifconfig pair2 description 'h1-eth0'
(repeat for pair3/s1-eth2 <-> pair4/h2-eth0)

*** Configuring hosts
ifconfig pair2 10.0.0.1/8 up
(repeat for pair4 at 10.0.0.2)

*** Starting controller
switchd -f /etc/switchd.mininet.conf -D ctl_ip=127.0.0.1 -D port=6653

*** Starting 1 switches
ifconfig switch0 create description 's1' up
ifconfig switch0 add pair1 add pair3
switchctl connect /dev/switch0

*** Starting CLI:
mininet>
```

Implementation: Multiple platform support

Nodes and Intfs per OS - "API" for OS-specific commands

- BaseNode
  - `getShell` : start host shell for a node
  - `popen` : run commands tied to a node
- BaseIntf
  - `makeIntfPair` : create virtual link endpoints
  - `moveIntfPair` : attach endpoints to nodes
  - `rename` : rename interfaces for book-keeping in topology

Implementation: Multiple platform support

Mid/high-level APIs largely untouched

- CLI, topology construction (Topo, Mininet) kept as-is
- `mn` untouched other than addition of new node types

```
$ doas ./test.py
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
mininet>
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=79277>
<Host h2: h2-eth0:10.0.0.2 pid=58592>
<IfSwitch s1: lo0:127.0.0.1,s1-eth1:None,s1-eth2:None pid=56473>
<Switchd c0: 127.0.0.1:6653 pid=92044>
mininet>
```

Implementation: Some weirdness

- the `ksh` prompt for root and `cmd()`
- visibility assumptions of a 'namespace'
- renaming interfaces
- topology startup order

Current status

Core features are done (barring bugs)

A longer list of to-dos...

- untested/unported:
  - MiniEdit
  - resource-limited links and nodes (cgroups, tc, iptables)
  - tons of example scripts
  - other controllers/vswitches?
- don't always run as root
- upstreaming...

Availability

- `net/mininet`, available since Aug 2017
- github fork (also with FreeBSD, Linux support): https://github.com/akoshibe/mininet

Acknowlegements

Special thanks to:

- Bob Lantz, Mininet developer
  for insight into Mininet and interest in having it ported,
- Reyk Flöter (reyk@)
  for introductions to switch and switchd and pointers to rdomains,
- Kazuya Goda (goda@)
  for insight into switchd's forwarding features,
- Peter Hessler (phessler@)
  for the crash course on port creation, mentorship, and suggesting this paper topic.

Questions?