Mininet on OpenBSD

Using rdomains for Interactive SDN Testing and Development

Ayaka Koshibe
akoshibe@openbsd.org

BSDCan 2018

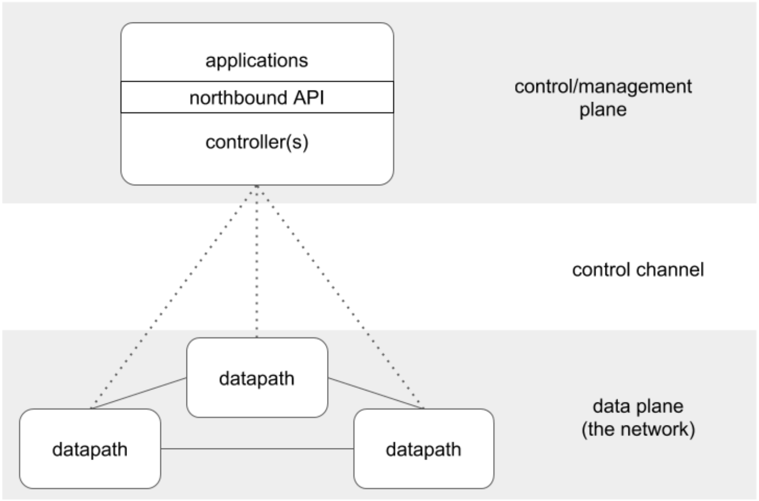
"SDN"?

Anything you want it to mean... Or rather,
a way to logically centralize control of network behavior

Instead of manually configuring each device:

- Separate packet handling elements from logic driving them
- Make packet handler(datapath) behavior controllable via API
- Program the datapath(s) from a control application(controller)
- Interact with the network via the controller's UIs/APIs

"SDN"?


applications
northbound API
controller(s)
control/management plane
control channel
datapath
datapath
datapath
data plane (the network)

OpenFlow

A control channel protocol standardized by the ONF

- Originally for Ethernet switches, significantly extended since
- 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
  - /dev/switchN : control channel for switchN
- switchd(8): controller
  - Implements flow forwarding and MAC learning logic
  - Can forward control messages to other controllers
- switchctl(8): control application for switchd(8)

SDN Stack Development

An SDN stack ***is*** a network - How do you test things?

- Hardware testbeds and labs
  - Resource-shared, limited customizability
- Dogfood on your own network
  - "Real" but careful treading required
- Models and emulations
  - Limited realism, but customizable and accessible

Mininet

An 'Emulator for rapid prototyping of Software Defined Networks'

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

Basic Usage: mn

Quick testing with built-in tests and components

```
beveren# mn --test=iperf
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1
*** Waiting for switches to connect
s1
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['1.13 Gbits/sec', '1.13 Gbits/sec']

(...teardown output)

#
```

Basic Usage: mn

Specify topologies, switches, controllers from sets of:

- Parameterizable topologies
- vswitches, controllers available alongside Mininet

```
# mn --topo=linear,3 --switch=sysbr --controller=none --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>
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()
```

Basic Usage: With an External Controller

Use a 'remote controller' to point a network to a running controller

mn:

```
# mn --controller=remote,ip=172.16.0.2,port=6633 --verbosity=output
 mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=24895>
<Host h2: h2-eth0:10.0.0.2 pid=24898>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=24904>
<RemoteController{'ip':'172.16.0.2', 'port':6633} c0: ... pid=24905>
```

Using the `add*()` API methods:

```
net = Mininet(topo=MinimalTopo())
net.addController(controller=RemoteController,
                  ip='172.16.0.2', port=6633)
net.start()
```

Development Workflow

For controllers/applications:

- Point a controller-less topology at running instance(s)
- Extend Mininet with a custom controller

For switches:

- Add a custom vswitch
- Wire up switch to a topology via a port on the Mininet host

...And if things go well, move to a real network

Development Workflow

(Somewhat) As an aside: For understanding SDN stacks

- Run self-contained and functional SDN networks
- Modify pre-packaged switches and controllers
- Compare networks with non-SDN networks/components

Internals: Core Mininet objects

- **Mininet**: coordinates the emulation process
- **Topo**: describes a (parameterizable) topology
- **Node**: represents/configures a single network node
  - **Host**: network-reachable end-host application
  - **Switch**: a network device/vswitch
  - **Controller**: controller application
- **Intf**: represents/configures a single network interface
  - **Link**: a pair of Intfs
- **CLI**: provides a CLI for the emulated network

Internals: Core Mininet objects on Linux

- **Node**: interactive `bash` running in a network namespace
  - Launched with `mnexec`, a wrapper around `setns` syscall
- **Switch**: OpenvSwitch instance
  - ofsoftswitch13(userspace switch), Linux bridge(non-SDN)...
- **Controller**: Stanford reference controller(`controller`) instance
  - Ryu, Pox, Nox...
- **Intf**: `veth` interfaces configured with `ifconfig`
- **Link**: `veth`s patched together with `iproute2`

Internals: Topology creation

Mininet: Basically some Python to run some commands

```
*** Adding (controller, hosts, 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>
```

Mininet on OpenBSD: Initial goals

- Recreate core features ("base" Mininet)
  - Parameterized and custom topologies with CLI
  - Built-in sanity tests
  - Run against external controllers
- Aim to eventually get it upstreamed
  - Preserve Linux support (for github fork)
- Reduce the number of external dependencies

Minimum requirements

- Network virtualization (separate address space), L2 and up
- Virtual interfaces, vswitches, and controllers for links and nodes
- Applications for baseline tests (ping, iperf)

rdomain(4) and pair(4)

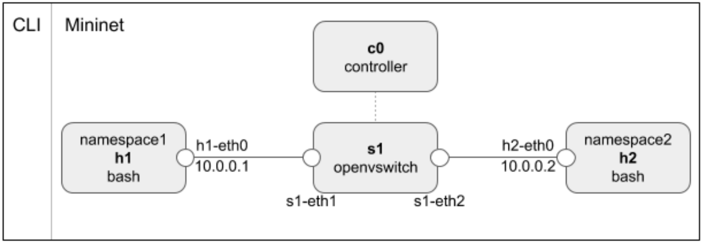
- A routing domain
  - Provides separate network address spaces
  - Recieves traffic via interfaces attached to them
  - Can restrict a process and descendants to its address space
- A pair(4) interface
  - Patched with another to form a virtual Ethernet link
  - Can be attached to an rdomain

Implementation: Mininet objects

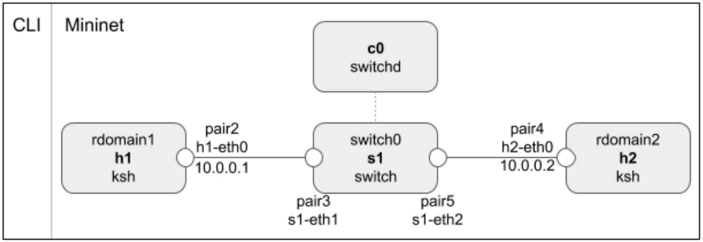
- **Node**: ksh running in a routing domain
  - Using route(8)'s `exec` command
- **Switch**: Node dedicated to a switch(4) instance
  - switchd(8) in forwarding mode for RemoteController case
  - bridge(4) for the non-SDN node type
- **Controller**: Node running switchd(8)
  - Uses Mininet-specific switchd.conf(5)
- **Intf**: pair(4) interface configured with ifconfig(8)
  - **Link**: Two patched `pairs`

A comparison

Original



OpenBSD



Topology creation revisited

```
*** Adding (controller, hosts, 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>
```

Multiple platform support

Node and Intf classes are tied to applications and commands

- Base* objects factored out into a "lowest" API
  - 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
- Nodes and Intfs derived from base classes for each OS

Multiple platform support

Mid/high-level APIs and `mn` largely untouched

- Basic topology scripts can be reused without modification
- `mn` untouched other than addition of new node types

```
# ./test.py
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> links
h1-eth0(pair2)<->s1-eth1(pair3) (OK OK)
h2-eth0(pair4)<->s1-eth2(pair5) (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

- Mininet's CLI and the `ksh` root prompt
- Visibility assumptions of a 'namespace'
- Interface names
- Startup order of objects in a topology
- Limit on number of rdomains to 255

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

Demo?

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 talk.

Questions?