Mininet on OpenBSD: Using rdomains for Interactive SDN Testing and Development

Ayaka Koshibe
akoshibe@openbsd.org

***Abstract*—Mininet is an interactive development tool designed for the purpose of prototyping and testing of Software-defined network (SDN) controllers, their applications, and SDN-capable switches. It, however, heavily depends on Linux-specific network virtualization features and applications. This paper describes the work to create a version of Mininet that is capable of running on OpenBSD by making use of rdomain(4)s and the SDN components available on it, namely switch(4) and switchd(8). Along the way, we describe the motivation for porting a tool like Mininet, and provide some examples of how it is used. We also describe some of the issues that were encountered in the porting process so far, and how they were resolved.**

1. Introduction

A developer of an SDN component can only get so far with piecemeal validation and unit testing. Sooner or later, that component will need to be tested within reasonably realistic SDN environments to see how it fares as part of a network. This points to the benefit of a development tool that can give developers easy access to customizable SDN-capable networks.

SDN developers tend to favor network emulation for its flexibility, footprint, and convenience. An emulated network can be quickly brought up and torn down, and take on arbitrary topologies incorporating the components being developed, in various ways. An emulator can also provide ways to run sanity checks against the networks that it creates. Simply put, an emulated network is able to model a sufficiently realistic network for a fraction of the cost, maintenance, and admininstration resources that would be required for a more realistic hardware-based network testbed.

This paper describes an SDN development tool called Mininet [1] and its OpenBSD port, net/mininet. Mininet allows developers to quickly create custom network topologies and test scenarios, and to interact with their emulated network via a CLI. However, it was also developed around Linux namespaces [2] and cgroups [3]. Although it saw usage during the development of OpenBSD's sflow controller, switchd(8), it had to be run in an Ubuntu VM [4]. The aim of this port is a version of Mininet capable of running natively on OpenBSD, to ease future development work around its SDN components.

2. Overview of SDN

2.1. Definition and Architecture

Since the term 'SDN' has many meanings attached to it, we first establish what we mean by 'SDN' in this paper. SDN is an approach in network design where a network is formed of two logical components, a control plane and a data plane. The former defines the logic that the latter follows in order to handle network traffic. The control plane, or controller, usually runs on commodity hardware with more processing capacity than the network hardware making up the data plane. The means used by the control and data planes to interact take many forms, with OpenFlow [5] being a well-known effort to develop a standardized protocol for this purpose. In practice, protocols such as SNMP and NETCONF, or RESTful APIs aren't uncommon, and it is possible to use a combination of different methods as long as the overall architecture is maintained.

One key result of SDN is the development of a global view of the network as a whole, so that its behavior can be managed as if it were a single entity rather than a collection of devices, each with their own and distinct interfaces. This unified view - an important basis for orchestration - is provided by the control plane on its northbound interface, in the form of some API. On top of the API are applications that control and collect network state, including UIs meant to replace the multiple terminals to each switch (or other SDN-managed device) with a single one to the control plane. Since the notion of a "northbound API" is not standardized to the extent of, e.g., OpenFlow, implementations and boundaries vary greatly between controllers and applications.

2.2. OpenFlow

While Mininet is designed to emulate SDNs in its general form, this paper focuses on SDNs based on OpenFlow since it is what the OpenBSD SDN implementation uses between the control and data planes. OpenFlow's name stems from its usage of packet header patterns to define different traffic classes, or flows, to which different actions are applied. These match-action pairs, or flow rules, are installed into the switch's flow table(s) via a control channel using the OpenFlow protocol. When a switch sees a packet that matches a flow rule, it applies the action to the packet.

Common actions include outputting it to some port or dropping it, rewriting a portion of the header, or searching for matches in another table.

Before flow installations can occur, the controller and switch must complete a handshake in order to establish the OpenFlow control channel. Among the information exchanged include information about the switch, such as its unique ID in the network (datapath ID), its port numbering, and capabilities.

What we have described here is a very high-level overview of OpenFlow; [5] provides the full specifications of OpenFlow 1.3.5, which is the protocol version implemented by OpenBSD.

2.3. OpenFlow on OpenBSD

The switch(4) [6] network interface pseudo-device implements a data plane node that supports OpenFlow 1.3.5. As expected from a data plane device, its forwarding behavior is defined by a controller. OpenBSD provides switchd(8) [7] as a controller to use with switch(4), and switchctl(8) [8] as a means to interact with the network through, and to control, switchd(8).

Some aspects of a switch(4) interface, such as its member interfaces, port numbering, and datapath ID, are not configured by switchd(8). These features are configured via ifconfig(8) and are announced by a switch during its handshake with the controller, or during statistics queries.

3. Mininet

3.1. Basic Usage

One key goal of Mininet is to be an interactive development platform for SDN components. It provides a command, `mn`, that allows a user to quickly launch prepackaged but parameterizable network topologies. `mn` also provides options for starting a CLI, specifying the components to use, and simple sanity tests.

For example, the following launches a linear topology of three switches, each with a host connected to it. The hosts will ping eachother in a 'pingall' test:

```
# mn --topo=linear,3 --test=pingall
(...startup output)
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
*** Stopping 1 controllers
(...teardown output)
```

When a test is specified, the topology is built before the test is run, and torn down immediately afterwards. This leaves little room for a closer look at the network. Another option is to launch a CLI connected with the topology. The CLI allows a user to run the same tests, but also inspect and manipulate nodes and links, and to run commands at network host shells. For example, the same test above can be run from a CLI, but also after a link is taken down and brought back up again:

```
# mn --topo=linear,3
(...startup 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> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

The `pingall` command in the CLI is functionally equivalent to running the ping(8) command from each host. Regular shell commands can be run from a Mininet host by appending the command after its name. For example, `h1 route` will cause host h1 to display its route tables.

Mininet also provides a Python API that allow developers to script custom topologies and test scenarios. A minimal one-switch, two-host topology with the CLI (what the `mn` command by itself will produce) can be recreated with a Python script:

```
#!/usr/bin/env python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI

class MinimalTopo(Topo):
    def build(self):
        # Add hosts and switches
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        s1 = self.addSwitch('s1')
        # Add links between nodes
        self.addLink(h1, s1)
        self.addLink(h2, s1)

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

Mininet has quite a few features, which are documented with its APIs at [1]. We will describe any relevant features as they appear in the discussion.

3.2. Components Development Workflow

Even by itself, Mininet has all components needed to create and run functional SDN-capable networks. Its flexibility as a development tool comes from how a developer can drop-in replace switches and/or controllers with their own, and use `mn` to quickly test their work with pre-canned tests or topologies. A developer can, for example, run their controller against a topology with loops using the `--topo=torus` option to test how well their cycle detection and forwarding works. Since controller/application development is considered a typical use case for Mininet, it also provides a `RemoteController` component that can be specified with the `--controller=remote` option. The `RemoteController` is an abstract controller object that points the switches to controller(s) already running elsewhere.

Although there are no means to incorporate external switches into the emulated network, Mininet's API allows developers to not only customize topology, but also to extend base network objects to create their own. These subclasses are recognized by Mininet as nodes and links that can be incorporated into toplogies, or, with little effort, have `mn` use it with its topologies and tests. As we will elaborate later, this API was useful in implementing the switchd(8) and switch(4) network objects for the port.

3.3. Original Implementation

We first describe the original implementation of Mininet before discussing how it was modified to work on OpenBSD.

3.3.1. Topology creation. A node or host in a Mininet topology is a shell - bash - running in its own Linux network namespace and assigned a unique name. A network namespace restricts network access of a process and its descendants to their own virtual network stack. Processes running in different network namespaces are allowed to interact through network links created by virtual ethernet (`veth`) pairs. The ends of a `veth` pair are either added to namespaces, or to switch nodes as member interfaces. A node might also be allowed to connect to the outside world by creating a link to a switch with outside connectivity.

3.3.2. Network objects. Since it is a shell, a node can easily be converted into a specialized component by having it run a few commands, such as launching a controller such as Nox or Ryu to become a 'controller node', or to create and configure an OpenvSwitch bridge to become a 'switch node'. A host, being a 'generic' node, does not require special initialization except for the shell. In terms of implementation, each node type is its own class and the child of the `Node` base class.

Links are created from the root namespace, since a link might join any two nodes. The ends are renamed to track their location in the topology - 'h1-eth0', for example, indicates the first interface on host h1. Mininet uses the `ip link` command from the `iproute2` utility collection in order to create, rename, and move `veth` link endpoints.

Mininet network objects define lifecycles, where each stage can be redefined through the 'low-level' API. This mechanism is used by, for example, the child classes of `Node` to run various commands at appropriate points in time to set up, configure, and tear down switches and controllers.

Now that the key details have been described, we can discuss the work done to get Mininet to run on OpenBSD.

4. Mininet on OpenBSD

As described in the previous sections, Mininet requires a host system to at least support network virtualization from layer 2 and above to construct topologies, and at least one supported virtual switch and controller for use in its canonical 'switch' and 'controller' nodes to implement topologies that support OpenFlow. switch(4), switchd(8), and switchctl(8), serve as the components for OpenFlow-capable nodes. rdomain(4) [9] and pair(4) [10] provide the virtualization needed for the topologies.

4.1. Overview of Routing Domains and pair(4)s

Routing domains, or rdomain(4)s, provide support for multiple in-kernel routing tables and address spaces. A process and its descendants can be restricted to using its own network address space and routing table by running it in an rdomain. Interfaces can be assigned to rdomains, through which processes in one rdomain can communicate with those in another. An interface of interest in our case is pair(4), a virtual Ethernet interface designed to be used in pairs, i.e. endpoints of an Ethernet link.

4.2. Initial Goals

In addition to recreating the core functions of Mininet, another goal was to minimize the number of external dependencies, which makes packaging simpler. This lead to some choices such as the use of ksh(1) to remove a dependency on bash.

The organization of the codebase also made it difficult to simply extend Mininet to support both Linux and OpenBSD. Part of the work involved quite a bit of refactoring to create a boundary between the node and link object implementations and the OS-specific network virtualization features. Once this was done, the APIs in Mininet were enough to implement the 'switchd' and 'switch' nodes, while retaining support for Linux.

Finally, sanity testing for Mininet involves the creation and destruction of interfaces and switches, making the testing process destructive. Because of this, we did not include regression tests for the package.

4.3. Implementation

In terms of functionality, rdomain(4)s and pair(4) interfaces are drop-in replacements for network namespaces and veth interfaces.

4.3.1. Nodes and Links. A node on OpenBSD becomes a ksh(1) instance running in an rdomain(4). Due to the way that a topology is initialized, nodes are created before links. To create a routing domain for the shell to run in, an extra interface is first created and assigned a rdomain before route(8) is used to run the shell in it. This means that nodes on OpenBSD will always have one extra interface.

Another quirk comes from how Mininet waits for a shell to return from a command. When starting a shell, it runs it interactively, with a shell prompt set to a sentinel character, and assumes that this sentinel is the only character in the prompt. Mininet detects that a shell has completed running the command when it sees this sentinel, returning its own CLI prompt and/or any command output assumed useful (e.g., the results of an iperf test). Due to requiring root to manipulate network namespaces, Mininet also assumes that it and everything else must run as root. The result was a modification of ksh(1) to make the '#' optional for root to prevent Mininet from waiting forever for the lone sentinel character.

A link is two patched pair(4)s where the endpoints may be members of a routing domain and/or a switch(4) instance. Mininet assumes that interfaces can be renamed as mentioned earlier, and also that the new name can be used to reference the interface on the system. Part of refactoring the lower-level parts of Mininet included adding a way for it to translate between its port names and those on the system. The same modifications were used for mapping Mininet-designated switch names (e.g. 's1') to the system's switch(4) interface names (e.g. 'switch0').

4.3.2. Controller and Switch nodes. The switchd(8)-based 'Switchd' node and switch(4) based 'IfSwitch' node are customized controller and switch Mininet objects, respectively, with initialization and teardown methods that run the needed `ifconfig` and `switchctl` commands. As the only OpenFlow nodes available on the OpenBSD port, these nodes are set as the default controller and switch types used by `mn` on OpenBSD. This means that the topology script described in Section 3.1, which is a typical level at which scripts are written for Mininet, can be used without modification. The only immediately noticeable difference would be the change in the network objects' types when inspection commands such as `dump` are run at the CLI:

```
# python ./test.py
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=28087>
<Host h2: h2-eth0:10.0.0.2 pid=46097>
<IfSwitch s1: lo0: ... pid=321533>
<Switchd c0: 127.0.0.1:6653 pid=31625>
```

In our case, `IfSwitch` replaces `OVSSwitch`, and `Switchd` replaces `Controller`. The PIDs shown are those of the ksh(1) instances that represent each node. These details would become visible to a user creating scripts written against the low-level API, where node and link classes or shell commands must be specified.

Unless `mn` is explicitly directed not to start a controller, an instance of switchd(8) will always be started. switchctl(8) can be used to configure switchd(8) to behave as an OpenFlow proxy between a switch(4) instance and another controller with the `forward-to` option when connecting the switch to the (local) switchd(8) instance. This feature is used to implement the `--controller=remote` option in `mn` by running switchd(8) as a proxy to the target.

4.3.3. Higher-level constructs. Since they are never exposed to the underlying system, the features of Mininet that rely on the 'mid-level' and 'high-level' APIs are not functionally affected by our changes. These features include the CLI and the topology object, which at most need to be aware of the existence of the newly created node and link objects. The topology may need to be able to create `IfSwitch`es, but do not need to understand how switch(4) interfaces are manipulated.

5. Current status

As of August 2017, Mininet has been made available in the OpenBSD ports tree as `net/mininet`. The current version of the port can be used to create and run OpenFlow-capable networks both as scripts and through `mn`. `mn`'s `ping` and `iperf` -based tests are also available, as well as 'baseline' bridge(4) nodes to create non-OpenFlow networks.

The imported version of the port is based on the fork of Mininet being maintained at

`https://github.com/akoshibe/mininet`

The repository's version of Mininet also includes support for FreeBSD with vnet jails and epair(4)s, using the same 'API' created when refactoring the node and link classes.

6. Future work

Due to the amount of features it has, and its reliance on Linux-specific applications, not all network objects are supported on OpenBSD, notably resource-limited hosts (which rely on Linux cgroups) and traffic-shaping links (which rely on `iptables` and `tc`). It also requires root in order to run, since it doesn't cleanly separate privileged commands from those that are not.

Additionally, we have yet to test some features, such as the MiniEdit GUI, and the 'cluster edition' remote links for interconnecting multiple Mininet network instances. We hope to gradually cover the unsupported or untested features, and to upstream our work. The latter may be a challenge due to the extensive changes that were needed in order to support multiple systems simultaneously.

7. Conclusion

SDN developers find appeal in network emulators for their ability to conveniently model SDN-capable networks, within which their components can be tested. Mininet is

one such tool that has seen usage in OpenFlow-based SDN development. We described how Mininet was ported to use rdomain(4)s, and in a way that it can be more easily ported to platforms with similar network resource virtualization features. The goal is that, by providing a version of Mininet that runs natively on OpenBSD, we provide a useful tool for further development of its SDN implementation.

Acknowledgments

Many thanks go out to Bob Lantz, for his work on Mininet and his interest in having it ported to systems outside of Linux, and for providing feedback; Reyk Flöter for suggesting an OpenBSD port of Mininet and introducing the author to switch(4) and switchd(4); Kazuya Goda for elucidating some of the design details of switchd(4); And finally, Peter Hessler, for helping the author import Mininet into the OpenBSD ports tree and for suggesting the author to write this paper.

References

[1] Mininet: An Instant Virtual Network on your Laptop (or other PC). [Online]. Available: http://mininet.org/.

[2] namespaces(7), namespaces - overview of Linux namespaces. Linux manual pages.

[3] cgroups(7), cgroups - Linux control groups. Linux manual pages.

[4] From conversation with Reyk Flöter, 9 Mar 17.

[5] Open Networking Foundation (ONF), OpenFlow Switch Specification, Version 1.3.5 (Protocol version 0x04), March 26, 2015.

[6] switch(4), switch - network switch pseudo device. OpenBSD manual pages.

[7] switchd(8), switchd - software-defined networking (SDN) sflow controller. OpenBSD manual pages.

[8] switchctl(8), switchctl - control the SDN flow controller. OpenBSD manual pages.

[9] rdomain(4), rtable, rdomain - routing tables and routing domains. OpenBSD manual pages.

[10] pair(4), pair - virtual Ethernet interface pair. OpenBSD manual pages.