Testing SDN behavior with Mininet

Sandbox Games

Mininet lets you test new controller features for your software-defined network in a sandbox before releasing them onto your production network. *By Philip Wette*

With a simple software update for the OpenFlow controller, a network admin can often change the entire behavior of an OpenFlow-based network [1]. You can even write the update yourself – after all, most controllers are available under open source licenses – but how do you find out whether your controller extension harmonizes with the topology of the production network? It would be too bad if you enabled your changes and the routing or the firewall went rogue. If you want to test the update extensively in a controlled environment first, you will turn to Mininet [2] sooner or later.

Sandbox for Network Admins

Mininet describes itself as "An instant virtual network on your laptop (or other PC)." With Mininet, you only need a single Linux system to emulate a network that deploys hundreds of virtual switches and hosts. Mininet can thus emulate a complete network with connected computers on a single machine; the host hardware decides the maximum number of switches and virtual hosts you can use. With an OpenFlow controller, you can then control this test network at will, and the emulated hosts will run any unmodified Linux program. Therefore, you can use Mininet to check whether and how the changed network affects individual programs running on it.

Mininet doesn't just test routing and forwarding rules. If you want to change the topology of a real network, you can simulate the consequences in advance using Mini-

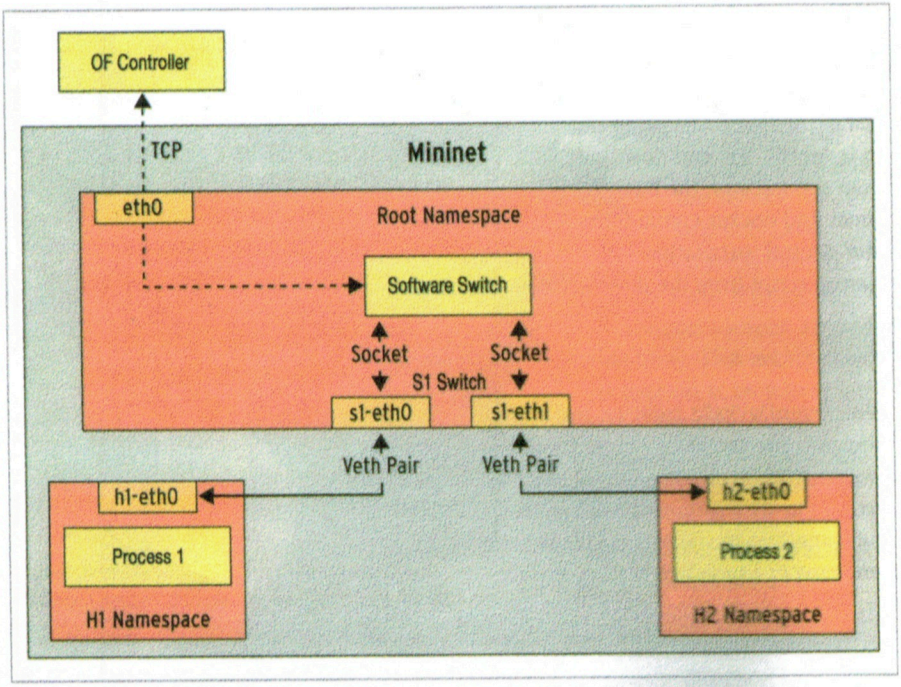

Figure 1: Mininet uses the kernel network namespaces to create separate virtual hosts and switches on a single host.

AUTHOR

Philip Wette is a Ph.D. student at the University of Paderborn, Germany, where he conducts research on the reconfiguration of network topologies through overlay networks.

net, or you can use Mininet to evaluate new networks in the planning phase.

Bag of Tricks

For Mininet to simulate a large number of virtual switches and hosts on a physical computer, it uses some sophisticated technologies that have been part of the Linux kernel for some time. For example, Mininet does without full virtualization and emulates virtual switches and hosts as simple processes on a shared host system.

Because these processes are to act like individual, networked devices, the kernel must separate them. The network namespaces [3] first introduced with Linux 2.2.24 are useful in this case. They allow you to equip processes with individual network interfaces, as well their own routing and ARP tables. Each process has its own network context; communication between two processes is handled by the virtual network interfaces assigned to them.

Processes, however, only talk to each other directly if a kind of virtual cable exists between their network interfaces. Mininet uses virtual Ethernet (veth) pairs to support the exchange of packets between two interfaces.

Figure 1 illustrates how the kernel uses network namespaces to equip the individual processes with their own network contexts. In this example, the two virtual hosts, H1 and H2, are connected to a common switch, S1. Bash processes emulate H1 and H2, the switch S1 runs in the root namespace in which the Linux kernel also operates. H1 and H2 use their own network namespaces and private network interfaces, `h1-eth0` and `h2-eth0`.

The switch S1 only has two ports, `s1-eth0` and `s1-eth1`, which use a veth pair to connect them to the corresponding host interfaces. Communication between H1 and H2 is thus exclusively via S1.

Packet forwarding between s1-eth0 and s1-eth1 is done by a software switch. It runs in the root namespace, which uses the physical interface `eth0`, and waits for commands from the OpenFlow controller. The controller typically runs outside of the Mininet host, often on another machine on the network.

Mininet also includes some controllers that can be installed using the installation script: Nox [4], the Open vSwitch controller [5], and the OpenFlow 1.0 reference controller [6]. The three run on the same physical machine as Mininet and talk to the switch via the local loopback device; they can be launched via the Mininet command line.

To assign specific properties to the emulated cables, Mininet talks to the Linux kernel's traffic-shaping tool `tc` (Traffic Control) [7]. In addition to the maximum data rate of each emulated link, the tool manages the packet error rate and latency. Additionally, you can use `tc` to define the buffering behavior of the network interfaces. This determines how the interfaces deal with packets in an overload situation. Besides a simple first-in, first-out (FIFO) method, more complex techniques such as Random Early Detection (RED) can be used here.

OpenFlow Versions

Generally speaking, Mininet can cope with various software switches, which in turn are compatible with different versions of OpenFlow. The current Mininet version 2.1 by default provides native support for the OpenFlow 1.0 reference switch [6], the Indigo Virtual Switch [8], and Open vSwitch [9]. The reference implementation is a pure userspace program, whereas the other two run as kernel modules and therefore achieve much better performance with a lower forwarding delay (Table 1).

All three switches implement OpenFlow 1.0. If you want to experiment with the new version, OpenFlow 1.3.*x* [10], you can replace the reference switch with the version from OpenFlow 1.3 (`ofsoftswitch13`). This is easiest to do if you install Mininet using

```
# mininet/util/install.sh -n3fx
```

TABLE 1: Supported Switches

Software Switch	OpenFlow Version	Mode
Reference implementation	1.0	Userspace
Indigo Virtual Switch	1.0	Kernel space
Open vSwitch	1.0	Kernel space
Ofsoftswitch 13	1.3	Userspace

Thanks to the `-n3fx` option, the network emulator provides an OpenFlow 1.3-compatible version of Nox.

Interaction with Mininet

Mininet offers an extensive Python API [11], which you can use to control the overall behavior of each component. You can also use it to define the topology of the network, launch arbitrary processes on hosts, change the parameters of a network interface dynamically, and cycle switch ports off and back on again during emulation. To observe the behavior of a network in a dynamic environment, Mininet even lets you add additional switches, hosts, and network interfaces during an ongoing emulation.

One special feature of Mininet is its interactive command-line (CLI) mode, which allows you to run commands on hosts while the emulation is running. Additionally, it lets you type and run Python code – for example, to change the network topology interactively.

To see whether the OpenFlow controller you are using can cope with dynamically appearing computers, the following example adds a host named `H3` to an existing network:

```
# py net.addHost("H3")

<Host H3: pid=1165>
```

In the next step, you patch the host to S1,

```
# py net.addLink( ➤
  net.get("S1"),net.get("H3"))➤
  <mininet.link.Link object at 0x13e1c90>
```

and enable the new interface on S1. To do this, you need to discover the name of the new interface:

```
# py net.get("S1").intfList()

[<Intf lo>, <Intf s1-eth1>, ➤
 <Intf s1-eth2>, <Intf s1-eth3>]
```

Armed with the knowledge that the name is `s1-eth3`, you can now enable the Mininet interface:

```
# py net.get("S1").attach("s1-eth3")
```

Finally, you need to configure the IP address on H3:

```
# py net.get('H3').cmd(➤
  "ifconfig h3-eth0  10.0.0.3")
```

A ping test finally checks to see whether H1 can reach the new host, H3. As Figure 2 shows, the test was successful. For a more detailed demonstration of the capabilities of Mininet, check out the OpenFlow article in this issue.

Conclusions

Mininet is ideal for implementing automated network experiments under realistic conditions. CLI mode makes it a handy tool for rapid prototyping development of controller extensions, and MiniEdit (Figure 3) provides a GUI for creating networks [12].

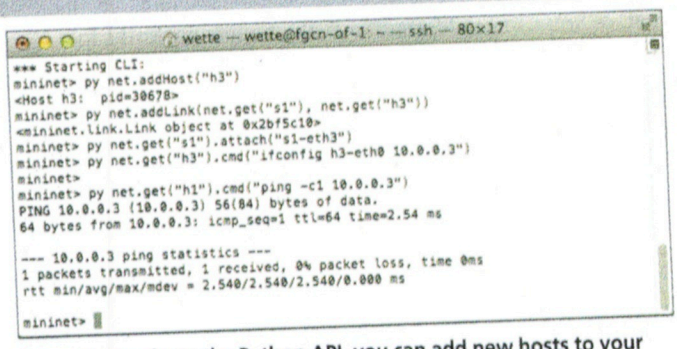

Figure 2: Thanks to the Python API, you can add new hosts to your Mininet with a handful of commands and configure the hosts in the same step.

Because Mininet only runs on one computer, its performance is limited. For example, on an i7 processor clocked at 3.2GHz, Mininet creates a total data throughput of 2.3Gbps. If you want to emulate a network with a higher data volume, you can try out the MaxiNet [13] project, which distributes Mininet across multiple physical computers. ■■■

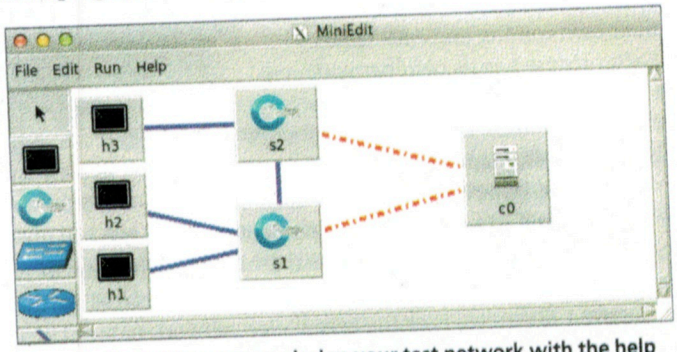

Figure 3: If you like, you can design your test network with the help of the MiniEdit graphical user interface.

INFO

[1] Mininet: *http://mininet.org*

[2] OpenFlow: *https://www.opennetworking.org*

[3] Network namespaces: *http://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/*

[4] Nox controller: *http://www.noxrepo.org*

[5] Open vSwitch controller: *http://openvswitch.org/cgi-bin/ovsman.cgi?page=utilities%2Fovs-controller.8*

[6] OpenFlow 1.0, release notes: *http://archive.openflow.org/wk/index.php/OpenFlow_1.0_release_notes*

[7] tc: *http://tldp.org/HOWTO/Traffic-Control-HOWTO/software.html#s-iproute2*

[8] Indigo virtual switch: *http://www.projectfloodlight.org/indigo-virtual-switch/*

[9] Open vSwitch: *http://openvswitch.org*

[10] OpenFlow specs: *https://www.opennetworking.org/sdn-resources/onf-specifications/openflow*

[11] Python API for Mininet: *http://mininet.org/api/annotated.html*

[12] MiniEdit: *http://gregorygee.wordpress.com/category/miniedit*

[13] MaxiNet: *http://www.cs.uni-paderborn.de/?maxinet*