A Mininet-based Virtual Testbed for Distributed SDN Development

Bob Lantz, Brian O'Connor
Open Networking Laboratory, Menlo Park, CA
lantz@onlab.us, bocon@onlab.us

ABSTRACT

The need for fault tolerance and scalability is leading to the development of distributed SDN operating systems and applications. But how can you develop such systems and applications reliably without access to an expensive testbed? We continue to observe SDN development practices using full system virtualization or heavyweight containers, increasing complexity and overhead while decreasing usability. We demonstrate a simpler and more efficient approach: using Mininet's cluster mode to easily deploy a virtual testbed of lightweight containers on a single machine, an ad hoc cluster, or a dedicated hardware testbed. By adding an open source, distributed network operating system such as ONOS, we can create a flexible and scalable open source development platform for distributed SDN system and application software development.

CCS Concepts

•**Networks** → **Network simulations;** •**Computing methodologies** → *Distributed simulation; Interactive simulation; Simulation environments;*

Keywords

Mininet; container based emulation; software defined networking; SDN; network OS; distributed systems

1. INTRODUCTION

The Fat Testbed Problem. Since 2013, we have observed that many users, developers, and testers of distributed SDN controllers and applications have constructed virtual testbeds using full machine virtualization and/or heavyweight containers, often for hosting the network OS or applications. This approach adds complexity and overhead, reducing both the usability and the scalability of the development platform.

In such a fat testbed, an additional (and, we argue, largely unnecessary) orchestration system may be required, such as OpenStack, Vagrant, or Docker. VMs and heavyweight containers consume significant hardware resources and frequently require complicated management. We believe there is a better way: using Mininet [1] to dynamically create virtual testbeds on the fly.

SIGCOMM '15 August 17-21, 2015, London, United Kingdom

ACM ISBN 978-1-4503-3542-3/15/08.

DOI: http://dx.doi.org/10.1145/2785956.2790030

2. A MININET-BASED VIRTUAL TESTBED

Mininet already has the capability of creating containers as well as the network from a single, simple Python API. Using Mininet avoids the need for installing, configuring and administering multiple orchestration systems. Moreover, Mininet hosts do not run unnecessary extra software such as multiple kernels or daemons, and they do not require unwieldy, bulky virtual file system images. To support configurations that exceed the resources of a single server, Mininet's experimental cluster support may be used to easily and seamlessly distribute the virtual testbed across multiple physical (or virtual) servers.

Just Enough Virtualization. By default, Mininet hosts are simply process groups connected to virtual Ethernet interfaces using Linux's network namespace feature. (Since Mininet 2.2, private mount namespaces have also been enabled by default, though this may become optional in the future.)

To make creation of a virtual testbed more convenient, we add a `Server` class which extends Mininet's `Host` class to enable slightly more virtualization on by default, including:

- Copy-on-write directories (e.g. `/var/run`, `/etc`) implemented using `overlayfs`
- A private PID space (PID namespace)
- Private `utmp/wtmp/btmp`
- Optional customized `hostname` (UTS namespace)

In addition, `Servers` can automatically run `sshd` for remote login, and we can run `dnsmasq` to provide DNS service for the virtual cluster if desired.

Most software can be run on unmodified Mininet hosts by simply using custom command line arguments (for example, specifying a local configuration file), but these changes make it easier and more convenient to deploy software onto the virtual testbed using the same scripts and configuration files which are used on a hardware testbed, without adding the computational, storage, and administrative overhead of virtual disk images, unnecessary daemons, or multiple kernels. Copy-on-write directories may either be temporary `tmpfs` file systems or may persist across runs as desired.

Cluster Support. Several distributed Mininet systems have been implemented, the most mature of them being Philip

Wette's Maxinet [2]. Since version 2.2, Mininet has itself included experimental support for simple execution on a cluster, including:

- A single console for control of a distributed experiment
- Support for the existing Mininet API with minimal alteration
- A simple `--cluster` option to create a distributed testbed
- Automatic node placement and tunnel creation
- Pluggable placement algorithms

Figure 1 shows a sample Mininet cluster, and figure 2 shows the code and/or commands which are used to create it.

Mininet's experimental cluster support requires minimal setup and configuration, making it well-suited for ad hoc networks such as a collection of student laptops or a cluster of physical or virtual machines that might only be available for a short time due to sharing or cost constraints.

A Mininet-based virtual testbed can flexibly support a number of use cases and run on a wide range of hardware configurations, anything from a single user's laptop to an ad hoc cluster of machines to a dedicated hardware testbed.

3. DEMONSTRATION

For our demonstration, we will show how an ad hoc cluster of rather ordinary computers (laptops and Raspberry Pi 2s) can be easily transformed into a distributed SDN testbed for development, research, experimentation, demonstration and other purposes. This simple testbed will contain hundreds of nodes running real application, switch, and network OS code – a configuration that is linearly larger in capacity than what Mininet supports on a single system. If time

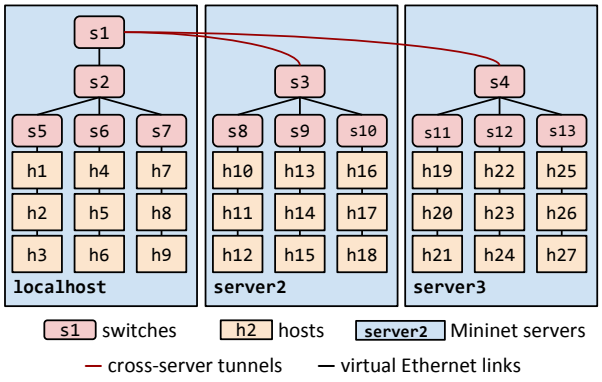

Figure 1: Sample Mininet Cluster

```
topo = Tree(depth=3, fanout=3)
servers = ['localhost','server2','server3']
net = MininetCluster(topo=topo, servers=servers)
net.start()
CLI(net)
net.stop()
```

...or via simple command line options:

```
# mn --topo tree,depth=3,fanout=3
     --cluster localhost,server2,server3
```

Figure 2: Mininet Cluster Python Code & Shell Command

and access permit, we may also add a scale-up demonstration of execution on a cloud or dedicated cluster as well as scale-down creation of a distributed testbed and development platform on a single machine.

We hope to demonstrate the convenience and speed of using Mininet and minimal, lightweight containers as a flexible and scalable platform for distributed SDN development, and to encourage the community to reserve more complicated virtualization for where it is actually required.

4. RELATED WORK

Several systems have been built to extend container based emulation to run on a cluster. Virtual Emulab [3] is a particularly notable example, and CORE [4] also allows cluster-based execution. Mininet has itself been extended to distributed execution in the Maxinet system, the Distributed OpenFlow Testbed [5], and Mininet's own experimental cluster support. Testbeds such as Emulab or DOT can be well-suited to research, teaching, or experimentation, but may introduce issues of contention over shared resources as well as configuration and administration. Maxinet and CORE have simpler installation and cluster configuration, and Mininet avoids configuration files, simplifying ad hoc clustering.

5. DISCUSSION: THE OPEN SOURCE, DISTRIBUTED SDN DEV STACK

By combining free, open source components, we can create a complete distributed SDN system and application software development stack. At the bottom, we have either a native Linux server, or a Linux virtual machine running on a VMM such as VirtualBox or KVM. On top of that, we use Mininet to create a virtual testbed consisting of a network of switches as well as virtual Linux servers for both end-user client/server and network OS applications. For our distributed control plane, we use a set of ONOS [6] instances that cooperate to provide fault tolerance as well as scale-out for capacity. Finally, on top of the ONOS instances, we can run single-instance applications or distributed applications which take advantage of ONOS's abstractions and APIs for managing shared state.

6. ACKNOWLEDGMENTS

Mininet core developers including Vimal Jeyakumar have built virtual testbeds using Mininet. Rich Lane submitted pull requests for mount and UTS namespaces. Brandon Schlinker also implemented mount and PID namespaces to support SDX. [7]

7. REFERENCES

[1] Mininet Project. Mininet. http://mininet.org/.
[2] P. Wette, M. Dräxler, A. Schwabe, F. Wallaschek, M. Zahraee, and H. Karl. MaxiNet: Distributed emulation of software-defined networks. In *IFIP Networking '14*, 2014.
[3] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the Emulab network testbed. In *USENIX ATC '08*, 2008.
[4] J. Ahrenholz, C. Danilov, T. Henderson, and J. Kim. CORE: A real-time network emulator. In *MILCOM 2008. IEEE*, 2008.
[5] A. R. Roy, M. F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba. DOT: Distributed openflow testbed. In *SIGCOMM '14*, 2014.
[6] ONOS Project. ONOS. http://onosproject.org/.
[7] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett. SDX: A software defined internet exchange. In *SIGCOMM '14*, 2014.