DOT: Distributed OpenFlow Testbed

Arup Raton Roy, Md. Faizul Bari, Mohamed Faten Zhani,
Reaz Ahmed, and Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo
{ar3roy, mfbari, mfzhani, r5ahmed, rboutaba}@uwaterloo.ca

Keywords

Testbed; Emulator; Software Defined Networking

1. INTRODUCTION

With the growing adoption of Software Defined Networking (SDN) technology, there is a compelling need for an SDN emulator that can facilitate experimenting with new SDN solutions. In this context, Mininet [1] has been proposed as an emulator for prototyping a network on a single machine. It allows users to create, control, and customize an emulated network on which they can run and test new control applications like routing, traffic engineering, *etc.* However, Mininet cannot scale for large networks and high traffic volumes [3].

To address these limitations, we developed *Distributed OpenFlow Testbed* (DOT), a highly scalable emulator for SDN. DOT distributes the emulated network across multiple physical machines to scale with network size and traffic volume. It also provides guaranteed compute and network resources for the emulated components (i.e., switches, hosts and links). Moreover, DOT can emulate a wider range of network services compared to other publicly available SDN emulators and simulators.

Our demonstration will illustrate several features of DOT including: (i) how easy it is to setup the emulator, (ii) how to deploy a topology using a single configuration file, (iii) how to run a connectivity test to ensure that the emulated network is properly deployed, and (iv) how to control and monitor the emulated components from a centralized location. We will also showcase DOT by emulating two applications: (i) policy based traffic steering through middleboxes [2] and (ii) traffic monitoring [5]. Our goal is to encourage researchers to use DOT for their SDN-related experiments and gather feedback to further improve its design and performance.

2. DESIGN

DOT provisions an emulated topology across a cluster of physical machines (Fig. 1(c)). One of these machines orchestrates the whole emulation process. We call this physical machine the `DOT Manager`. The other machines are called `DOT Nodes`. These machines contain virtual switches and hosts used to build the emulated topology.

Our embedding algorithm (detailed in [4]) considers the resource capacity of each `DOT node` and finds an embedding that minimizes physical bandwidth usage and the number of physical machines. The embedding algorithm partitions the emulated network and maps each partition to a particular physical machine. When the algorithm embeds virtual links, two cases may arise due to partitioning. First, if the virtual link connects two virtual switches embedded in the same physical machine, then this link is provisioned inside that machine (link `b` in Fig. 1(a)), and hence we call it an *intra-host virtual link*. Second, a virtual link connecting two virtual switches residing at different physical machines (hereafter called a *cross-host virtual link*) is mapped onto a path passing through the physical network (link `e` in Fig. 1(a)).

An intra-host virtual link is emulated by instantiating two virtual Ethernet interfaces (`veth(s)`) within the same physical machine (using Linux `add ip link` command). The virtual link bandwidth and delay are emulated using the `tc` and `netem` commands, respectively.

To forward traffic passing through a cross-host virtual link, we create a specialized switch called the *Gateway Switch* (GS) in each physical machine (`GS1` and `GS2` in Fig. 1(b)). GSs are connected through pair-wise `GRE` tunnels. Then we create two segments (using Linux `add ip link` command) for a cross-host link in each physical machine where its end points reside (in Fig. 1(b), `e` has two segments e' and e''). Each segment connects one of its end points to the GS in the physical machine. A unique identifier is assigned to each cross-host link, *i.e.*, both segments have the same identifier.

Whenever a GS receives a packet from one of the segments, it encapsulates the packet within a `GRE` header, tags this packet with the identifier of the segment, and unicasts it to the destination physical machine. The receiving GS (at the other end of the tunnel) decapsulates the packet, inspects the tag, and forwards the packet to the corresponding segment. We pre-install forwarding rules in the GSs to enable this mechanism. It is worth noting that GSs are completely transparent to the DOT users and are used to hide the fact that different portions of the emulated network are distributed across a cluster of physical machines. Moreover, DOT does not install any forwarding rules in the virtual switches of the emulated topology.

For emulating the end hosts, DOT supports both OS level (`Linux Container`) and full virtualization (`Kernel Virtual`

SIGCOMM'14, August 17–22 2014, Chicago, IL, USA
ACM 978-1-4503-2836-4/14/08.
http://dx.doi.org/10.1145/2619239.2631457.

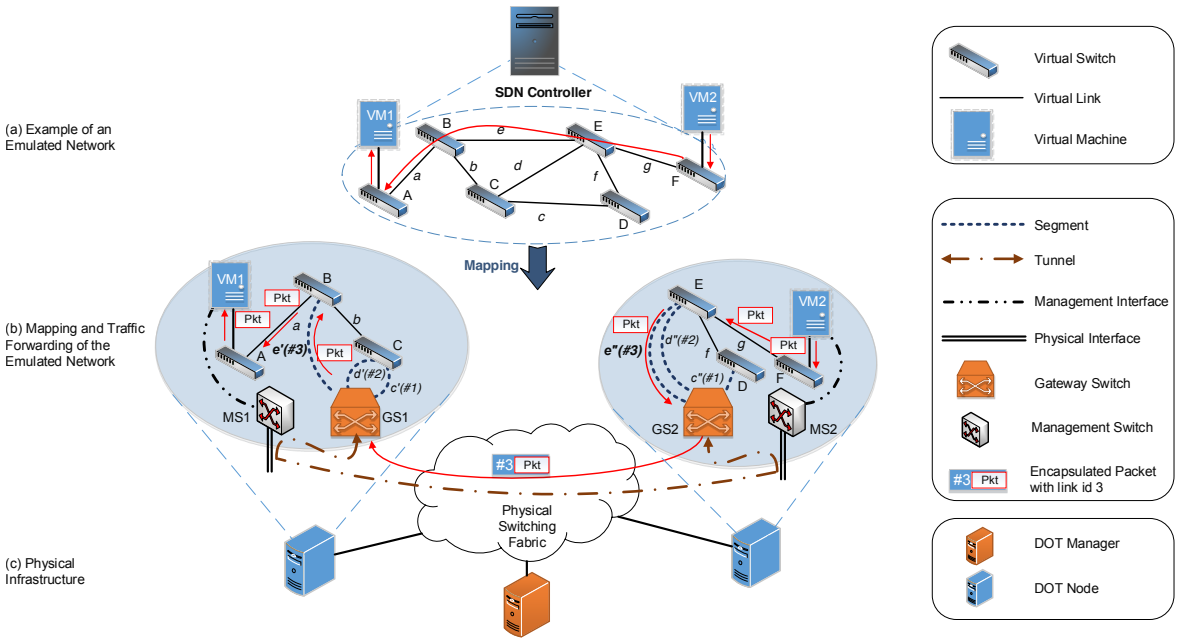

Figure 1: Emulated network embedding and traffic forwarding in DOT

`Machine (KVM)`). Moreover, `tiny core Linux` (a minimalistic flavor of Linux) is used as guest OS to minimize resource footprint of the Virtual Machines (VMs).

An emulated network in DOT also includes a separate (out-of-band) management network. Each VM/container is equipped with a virtual management interface. This interface is attached to a management switch (`MS1` and `MS2` in Fig. 1(b)) of the respective physical host. The management network allows direct access (using `ssh`) to the VMs/containers and ensures that management traffic does not interfere with emulation traffic in any way to affect the outcome of an experiment.

3. FEATURES

The key features of DOT are as follows:

- **Scalability:** DOT can be deployed across multiple physical machines. It can be scaled to meet the requirements of the emulated topology. This feature provides DOT an edge over other emulators.
- **Transparency:** The specialized components required for distributed deployment are kept hidden from the users' controller(s). As a result, intra-host and cross-host links appear indistinguishable to the controller(s).
- **Guaranteed performance:** Our embedding algorithm provides resource guarantee for all emulated components (*i.e.*, switches, links, and hosts). Thus, the performance of the emulated network is close to that of a real network.
- **Flexibility:** DOT supports both OS-level and full virtualization. Full virtualization enables DOT to use heterogeneous (different versions of kernel) end hosts. It also provides opportunities to emulate a larger class of network services, *e.g.*, managing middleboxes [2].
- **Multi-tenancy:** DOT can be easily extended to provide access to multiple users to run simultaneously their emulations on the same physical infrastructure.
- **Hybrid networks:** Non-OpenFlow switches can be deployed in DOT. A user can use DOT for emulating a network that includes both OpenFlow and non-OpenFlow switches.
- **Physical switch integration:** Physical switches can be included in an emulated network in DOT. This feature enables flexible experimental setup to aid gradual rollout of SDN technology in the production network by emulating one part of the network in DOT while keeping the other parts running on actual hardware.

4. IMPLEMENTATION

DOT is available as an open source software at `www.dothub.org`. We have deployed networks with different sizes and resource requirements to test our current implementation. Almost all publicly available OpenFlow controller can be used with DOT. We have tested DOT with Floodlight, OpenDaylight, NOX, and POX.

5. ACKNOWLEDGMENTS

This work was supported by the Natural Science and Engineering Council of Canada (NSERC) under the Smart Applications on Virtual Infrastructure (SAVI) Research Network.

6. REFERENCES

[1] Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., and McKeown, N. Reproducible network experiments using container-based emulation. In *ACM CoNEXT 2012*.

[2] Qazi, Z. A., Tu, C.-C., Chiang, L., Miao, R., Sekar, V., and Yu, M. SIMPLE-fying middlebox policy enforcement using SDN. In *ACM SIGCOMM 2013*.

[3] Roy, A., Yocum, K., and Snoeren, A. C. Challenges in the emulation of large scale software defined networks. In *APSYS 2013*.

[4] Roy, A. R., Bari, M. F., Zhani, M. F., Ahmed, R., and Boutaba, R. Design and Management of DOT: A Distributed OpenFlow Testbed. In *IEEE/IFIP NOMS 2014*.

[5] Yu, M., Jose, L., and Miao, R. Software defined traffic measurement with OpenSketch. In *USENIX NSDI 2013*.