Section 4: Week 8: Experiment with Mininet

Nate Bachmeier

TIM-7010: Computer Networking and Mobile Computing

September 18th, 2019

North Central University

# Experimentation with Mininet

Software-defined networking represents the next evolutionary step in device communication (Eissa, Bozed, & Younis, 2019). It enforces clear separation of duties between the control plane and the data plane.

The control plane is responsible for making decisions around flow control and is extensible through modules registering to network events (Azodolmolky, 2013). For instance, a simple learning switch can monitor for incoming packets and build a map of sender Media Access Control (MAC) to the physical switch port. Without this map, incoming traffic needs to be broadcasted to all physical switch ports, to ensure it arrives at the correct location (Dordal, 2005).

The data plane consists of simple packet forwarding switches and possesses the least amount of logic possible (Azodolmolky, 2013). The de-facto protocol used by these switches is called OpenFlow. OpenFlow leverages flow tables to perform match operations and then perform one or more actions on the incoming traffic. For instance, the flow table entry might say drop any traffic for network port 1234. Another entry could forward virtual endpoints, such as Named Data Network (NDN) resources, to a contextually dependent Internet Protocol (IP) address (Lei, Zong, Zhu, & Zhang, 2018).

To explore these concepts, several researchers use a tool called mininet (Santos de Oliveira & Shinoda, 2014). The utility provides a scriptable interface for defining detailed network topologies and then materializing them as lightweight virtualized endpoints.

The materialization uses Linux’s Network Namespace feature to associate different routing tables with each launched application instances. For example, mininet could start two copies of the Secure Shell Daemon (sshd) and bind them to addresses 10.10.10.10 and 20.20.20.20 (Lantz, 2016). During binding, the kernel will create separate virtual network adapters and treat them as if they were different physical machines. This behavior enables technology practitioners to experiment on large topologies using a single server.

Topologies can be significantly more complicated by combining multiple virtual switches to represent different network segments (Pal, Veena, Rustagi, & Murthy, 2014). Internally the mininet.topo.MultiGraph supports adding arbitrary nodes and edges. A node represents a virtual endpoint and uses a Python class to define its implementation. The edges expresse the routing table configuration.

For example, a Python class can derive from the mininet.node.Node base class and define an initialization script to become a Linux router (Lintz, 2016). Then one or more routers can be added into the topology, each with custom local configuration.

Links from virtual switches can connect to the router and other topology nodes. Each of these links can contain configuration parameters, such as latency or fault injection. These capabilities allow researchers to verify reliability scenarios that are otherwise difficult to reproduce.