Educational Virtual Routing Labs with IPMininet

Mathieu Jadin
UCLouvain, Belgium
mathieu.jadin@uclouvain.be

Olivier Tilmans
Nokia Bell Labs, Belgium
olivier.tilmans@nokia-bell-labs.com

Maxime Mawait
UCLouvain, Belgium
maxime.mawait@student.uclouvain.be

Olivier Bonaventure
UCLouvain, Belgium
olivier.bonaventure@uclouvain.be

ABSTRACT

Learning computer networks requires to understand the interactions between various protocols and layers. While using virtual labs in networking courses is key to enable students to master their understanding, their use is often restricted to advanced courses or vendor certifications due to their complex setup. We introduce IPMininet, a virtual lab framework built on top of Mininet, as a lightweight and practical way to enable students to experiment with virtual IP network topologies. By abstracting protocol configuration and daemon management, IPMininet enables even students with no networking background to use virtual networks while speeding up exercise development for teaching staffs. We illustrate some of IPMininet's features using a short example that spawns a virtual lab with custom AS-peering relationships, internal traffic engineering, firewall, and DNS.

1 THE NEED FOR VIRTUAL LABS

Students typically learn computer networks through classes and exercises on paper with tutors. As networks consist of several layers and protocols, understanding their interactions and the associated concepts is difficult without hands-on experience. To address this, some courses complement traditional classrooms with a few practical hands-on sessions, using real protocol implementations [13]. These lab sessions however often require a significant time investment from the teaching staff, mostly spent on equipment setup (e.g., router configuration, IP addressing).

This is insufficient for at least two reasons. First, this setup complexity often constraints the use of practical labs to a few "larger and more important" topics—e.g., BGP peerings. Students experiencing difficulties with any other part of the course material (e.g., the longest prefix match in IP routing) will not benefit from such labs, and at the same time are likely to be stuck with the few written exercises predefined in the course. Second, the skills required to alter these labs can present a barrier to student experimentation—e.g., changing an IGP weight requires to identify the right configuration file, make the change without causing a parsing error, and figure out how to restart the routing daemon. Additionally, the boilerplate present in these lab configurations tends to hide the high-level concepts behind the lab, making it harder to grasp for students—e.g., demonstrating the effect of IGP link weights on the routes still requires configuring router IDs, assigning IP prefixes to links, etc.

In addition to the above challenges, the number of students enrolled in networking courses is larger than ever. While automated grading platforms can help to keep providing personal feedback to students [4], this makes it impossible to use physical labs both for cost and flexibility reasons (e.g., unplugging cables in-between experiments). Instead, networking labs have to be virtualized. We note that this is also the approach chosen by equipment vendors for their different certifications [6, 14, 21]. While virtualization brings tremendous flexibility (e.g., rebooting the whole lab is a matter of seconds, hundreds of nodes can be virtualized on one machine), it also brings new challenges of its own (e.g., multiple devices share the same screen and are identified only by their CLI prompt label, troubleshooting students' labs is often more challenging as those are local to their machines).

To address the above challenges, we have developed IPMininet, a framework built on top of Mininet [16], which enables to quickly define virtual labs. In [4], the authors report on their experience with open educational resources including IPMininet to support computer science education. This paper is focused on IPMininet itself. More precisely, we present in §2 how IPMininet provides abstractions for the most common components in IP networks, enabling to quickly design virtual labs focused on key concepts, and easily distributing them to students. We illustrate this in §3 by describing a simple virtual lab which showcases key IPMininet features. We then discuss in §4 other environments that enable "hands-on" experiments, and conclude in §5.

2 IPMININET ENABLES LEAN VIRTUAL LABS

Open-source frameworks, such as Netkit [23] and Mininet [16], enable to develop virtual labs that have two advantages: *i)* they use an intuitive format to describe virtual lab topologies; and *ii)* hide away most of the complexity involved in creating the virtual topology. In particular, by leveraging Linux namespaces, Mininet is able to instantiate several hosts, switches and routers, on a single host with nearly no overhead, as all virtual nodes share the same kernel.

Both frameworks however provide limited functionality beyond spawning a virtual network. Indeed, they expect their users to manually configure all properties beside layer-2 (i.e., Ethernet MACs), such as the daemons to run, their configurations, the IP addresses and allocations, ... While some configuration properties will vary across experiments (e.g., IGP weights), most of these can be considered as boilerplate (e.g., IP address allocation). Worse, by forcing the lab designer (e.g., the teaching staff) to provide full-blown configuration files for all daemons in their respective dialects, it deeply buries the focus points of the virtual lab, often preventing the users (e.g., students) from identifying the main goal of the lab. For example, as

Mininet is designed to enable SDN (e.g., OpenFlow) experimentation, its API is centered around building a layer-2 switched network (i.e., all the IP addresses it assigns are part of the same prefix). Using it to virtualize IP networks then requires the experimenter to write down in the topology description all IP-layer properties, and provide daemon-specific configuration files for each node [24].

IPMininet [25] is a set of Python classes extending Mininet which provides a declarative API to instantiate IP networks, abstracting away the configuration of the individual components (daemons, interface settings,...). In other words, IPMininet enables the experimenter to express the key properties for a particular lab, as part of the topology description process, and then automatically derives all missing properties before generating all daemon-specific configuration files (in their respective language). This ensures that the resulting virtual lab description files stay as focused as possible, e.g., enabling the students to change the labs "by example", i.e., using the presented APIs in a single file.

In order to streamline the distribution of virtual labs to students, IPMininet comes with an integrated build script which takes care of installing all components for which it provides an abstraction on the host system or a virtual machine[1]. This ensures that the students' machine supports all the features provided by IPMininet, and minimizes the administrative burden for the student, which is particularly important for introductory-level courses. From there professors can then share virtual labs as self-contained files, which can for example be part of a classroom's shared git repository (thus automating further the lab distribution process). Once the lab files are downloaded, students can then work without Internet connectivity.

We released IPMininet as well as sample labs as Open Educational Resources (see §A). At the time of writing, IPMininet provides abstractions for FRRouting [1] IP routing protocols (OSPF(v6), RIPng, BGP, PIM), Facebook's OpenR [11], IPv6 Segment Routing, IPTables, Radvd, BIND9, sshd,... In particular, the BGP abstraction enables to define peering relationships between ASes, iBGP meshes, route reflectors, all aimed towards facilitating the understanding of their effect by students. IPMininet has 7376 python code lines: 2570 lines for automated tests and 1739 lines defining 37 examples. It also has 401 lines for the configuration templates and a sphinx documentation of 1660 lines.

3 IPMININET IN PRACTICE

We now comment a small virtual lab aimed to illustrate IPMininet, whose topology is visible in Figure 1 and its code in Listing 1. More precisely, we start from a network containing two hosts connected to a switch using Mininet's classic API (lines 8 to 10). We then gradually introduce features from IPMininet, linking the key concepts that the lab manipulates to its implementation, and discussing how students can perform experiments to understand it.

3.1 Introducing IP prefixes

Concepts Before teaching routing to students, they need to understand addressing. Each part of the network, a LAN, has one or more IPv4 and/or IPv6 addresses assigned that are used to contact them. And we also introduce routers to showcase that they route traffic through LANs by looking at their routing tables. To keep things simple for them, we start with static addressing and static routes.

Implementation IPMininet provides a method to quickly define a router: `addRouter()`. It also adds support for IPv6 addressing. By default, IPMininet auto-allocates one subnet for both IPv4 and IPv6 for each LAN with routers and hosts, unless they were already specified in the topology description.

Moreover, hosts have a default route towards their access router in both IPv4 and IPv6. To insert static routes, we use the STATIC daemon of FRRouting [1] to set up static routes.

Experiments These few lines let students experiment with IP routing. Indeed, as IPMininet auto-allocated IP ranges and addresses, it also filled routing tables. Consequently, students can learn to inspect routing tables, and test reachability of addresses from hosts connected using a different prefix using pings.

3.2 Dynamic intra-domain routing with OSPF

Concepts With previous features, students can understand how packets are forwarded through the network: by following routing tables on each router. They need to understand how routing can be resilient to failures. Network operators typically use link-state routing, for instance, OSPF for IPv4 and OSPFv3 for IPv6. These protocols install routes for each destination prefix by computing the shortest-paths on the network graph. Students also need to understand that we can influence this computation with link weights. For instance, a satellite link is less interesting to use if a faster alternative exists.

Implementation IPMininet supports the OSPF and OSPF6 daemons of FRRouting [1]. Given their popularity in the real-world, IPMininet enables by default OSPF and OSPF6 on each router. This can be obviously changed, e.g., to design labs using RIPng.

IPMininet adds a parameter `igp_metric` to the `addLink` call (line 19). This parameter is passed to the OSPF and OSPF6 daemon classes when they generate their configuration files, in order to control the resulting network paths.

Experiments Beside connectivity checks using pings, and dumping routing tables, students can start to use traceroute to observe the paths computed by OSPF.

A natural follow-up is then to ask them to tune the IGP metrics on the different links, e.g., to arrive to a desired set of forwarding paths.

In parallel, students can also bring links up and down using the Mininet CLI, and observe the reaction of the routing protocols. A more comprehensive exercise can be to maintain some forwarding property under some failure condition, by optimizing the weights. By having access to a live network, and easily toggle link states, students can then individually find a solution and verify it.

3.3 Connecting networks with BGP

Concepts BGP is more complex to understand and configure than OSPF. Most BGP configurations are hard to read and even harder to debug. Students first learn external BGP, and we present them two potential peerings: *i)* the client-provider peerings in which an AS, the client, buys connectivity from another AS, the provider; *ii)*

[1] It also provides scripts to create a new virtual machine on which IPMininet will be installed

the shared-cost peering in which two ASes choose to share the cost of the BGP peering. They also see the local pref parameter and the importance of the longest match. In advanced networking classes, we introduce them to iBGP sessions [18]. Only then, they need to play with more advanced concepts like MED, BGP communities, and route reflectors.

Implementation IPMininet supports the BGP daemon of FRRouting [1]. For the same reason as OSPF and OSPF6, IPMininet provide a configuration object, called `BorderRouterConfig` (see Line 43), enabling BGP, OSPF and OSPF6 on the router with appropriate default parameters. Students do not have to bother with redistribution settings when exploring BGP. We can declare the AS in which routers are by using the parameter `asn` when adding the router. We declare new ASes through Line 24 to Line 26.

After setting the AS, we define the external peerings with the `ebgp_session` call. An optional argument is used to set the predefined types of peering. We define a shared-cost peering on Line 28 and a client-provider one on Line 31. Behind the scene, IPMininet uses local pref parameters and BGP communities.

To declare iBPG fullmeshes, we do not need to add every peering manually, we use the `addiBGPFullMesh` call. This also sets the AS number of the routers in the mesh (see Line 22).

Experiments As for the previous daemons, students can check connectivity and paths by using pings and traceroutes, or by dumping the routing tables. Student also have access through the FRRouting CLI to the full BGP peering state. Thus, they can check the BGP route decision process.

For instance, we can ask students to find out which AS does not have connectivity to a given prefix and which peerings it needs to establish in order to guarantee connectivity.

They can also bring some links up and down, and observe the reaction of the protocol. BGP, unlike OSPF, can converge to different routing tables when taking a link down and then back up again.

They can also play directly with the local pref parameter or change the length of the prefix to force traffic to go through a given AS.

3.4 Traffic steering with IPv6 Segment Routing

Concepts IPv6 Segment Routing [8] is a modern version of source routing, available since Linux 4.10 [17]. It uses an extension header of IPv6 to steer traffic thanks to a given list of IPv6 addresses, called segment. A segment can specify an intermediate node to pass through, for instance a firewall. It can even specify network functions to apply to packet. Students learn this extension in the advanced networking class.

Implementation IPMininet offers a series of abstractions to insert or react to IPv6 Segment Routing Headers. Line 47 showcases `SRv6Encap` that inserts an IPv6 Segment Routing Header to each packet going through r1 with destination h3. This header steers the traffic through r2, even if that's the highest cost path.

We use the `post_build` method because this method is called after the IP address auto-allocations, and we need that info to build the IPv6 Segment Routing encapsulations.

Experiments The main use of IPv6 Segment Routing is to steer packets through a particular path. Students can observe the path taken with and without the routes through traceroutes and routing table dumps. For instance, we can ask students to produce the encapsulation rule to forward traffic through a given path.

3.5 Building a DNS hierarchy

Concepts In practice, most people do not encode IPv4 or IPv6 addresses by hand. So students need to understand DNS resolution. They need to grasp the domain name hierarchy and the DNS queries that are sent to resolve a single domain name. They also need to understand how they can influence DNS caching and even perform rough load balancing between servers by tweaking DNS records.

We also introduce the master-slave replication of DNS servers to show them how a service can stay almost always up.

Implementation IPMininet offers a series of abstractions to the Named daemon of bind9. To work properly, a DNS zone at least needs to have, on each DNS server, a SOA record, a few NS records before adding the A and AAAA records. All of this configuration is entailed by a `addDNSZone` call. For instance, Line 36 defines a DNS zone for AS1.

Experiments The best way to understand the DNS architecture is to manually query each DNS domain in the chain, as a DNS resolver would do it. They can use the dig utility for that.

4 RELATED WORK

IPMininet is not the only available resource to build virtual labs to support networking education.

Virtual networks CORE [2], GNS3 [9], Netkit [23], and Mininet [16], all define ways to virtualize networks of multiples nodes and links within a single machine. By virtualizing production-grade protocol implementation, they allow students to gain experience with real systems. While powerful, their use for teaching labs require to manually configure all IP-layer and above aspects of a lab (e.g., interface addressing, protocol configuration).

Network simulators ns2 [10], ns3 [12], and omnet++ [26] define a programming framework to realize discrete-event simulations of networks. These are complementary to IPMininet in that they enable to precisely test specific behaviors (e.g., TCP congestion control). These are however only models (e.g., could diverge from alternative implementations), and do not operate in real-time.

Configuration generation Autonetkit [15] (and its commercial counterpart, VIRL [22]), share a common goal with IPMininet: Automating configuration generation based on some high-level lab description. Both of these are however focused on in-depth routing protocol configuration, thus lacking the support for e.g., iptables, bind9, and tend to be more difficult to use for students due to their very detailed approach. Configuration synthesis is a very active research area [3, 7, 19, 20], which aims to define network configurations from the specified *SLAs (end-goals)*, whereas IPMininet aims to automate the generation process itself.

5 CONCLUSION

IPMininet is a modular, well-tested, framework which enables to virtualize complex IP networks and functions by extending Mininet. We released IPMininet and several labs as open-source software, and welcome any contribution to improve the API, add support for other daemons, or define new virtual labs.

```python
from ipmininet.utils import otherIntf
from ipmininet.iptopo import IPTopo
from ipmininet.router.config import *

class PaperTopo(IPTopo):
    def build(self, *args, **kwargs):
        # Classic Mininet API
        h1, h2, s1 = self.addHost('h1'), self.addHost('h2'), self.addSwitch('s1')
        self.addLink(h1, s1)
        self.addLink(h2, s1)

        # IPMininet APIs from now on
        r1, r2, r3 = self.addRouters('r1', 'r2', 'r3')
        h3 = self.addHost('h3')
        # Handy shortcut for multiple addLink() calls
        self.addLinks((r1, r3), (r1, s1), (r2, s1), (r3, h3))
        # Control the IGP shortest-path computation to favor s1-r1-r3-h3 over s1-r2-r3-h3
        # due to the lower path cost
        self.addLink(r2, r3, igp_metric=5)

        # Define all 2 routers as being in an iBGP fullmesh in AS1
        self.addiBGPFullMesh(1, (r1, r2, r3))
        # Add satellite ASes
        as2, as3 = self.addRouter('as2', asn=2), self.addRouter('as3', asn=3)
        as4      = self.addRouter('as4', asn=4)
        self.addLinks((r1, as2), (r1, as3), (r1, as4))
        # SHARE peering type setup export filters excluding routes learned from providers
        ebgp_session(self, r1, as2, link_type=SHARE)
        ebgp_session(self, r1, as3, link_type=SHARE)
        # Register AS4 as provider for AS1, i.e., AS2/AS3 cannot access it through AS1
        ebgp_session(self, r1, as4, link_type=CLIENT_PROVIDER)

        # Create a new TLD (aptly named tld.) spanning AS1 with a master-slave replication
        h1.addDaemon(Named)
        h2.addDaemon(Named)
        self.addDNSZone(name="tld.", dns_master=h1, dns_slaves=[h2],
        nodes=[h1, h2, h3, r1, r2, r3])

        super().build(*args, **kwargs)

    def addRouter(self, name, **kwargs):
        # Replace the default from OSPF routers to OSPF + BGP
        return super().addRouter(name, config=BorderRouterConfig)

    def post_build(self, net):
        # Tunnel IPv6 traffic crossing r1 towards h3 through r2
        SRv6Encap(net=net, node='r1', to='h3', through=['r2'], mode=SRv6Encap.ENCAP)
        super().post_build(net)
```

Listing 1: IPMininet provides powerful APIs (in red) to create virtual labs for IP networks.

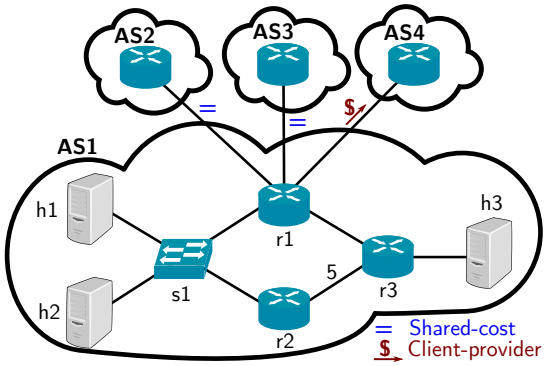

Figure 1: Listing 1 creates this toplogy. All links have an IGP weight of 1, except for the link between r2 and r3 with an IGP weight of 5.

IPMininet enables experimenters, students and teachers alike, to quickly build virtual labs to test IP network concepts. IPMininet defines a high-level API to easily specify network-wide behaviors, and automatically generate the corresponding configurations for all network nodes. By automating the boilerplate traditionally found in most virtual labs, and hiding it from the lab descriptions, IPMininet enables students to focus on the key concepts for each virtual lab.

A OPEN EDUCATIONAL RESOURCES

IPMininet is publicly available at https://github.com/cnp3/ipmininet under the GPL-2.0 license. The repository contains numerous virtual labs, examples for all daemons, and links to its API doc. We welcome all contributions through pull requests. To validate contributions, IPMininet automated tests are run on a Jenkins server. Bug reporting is possible through issues and the mailing list ipmininet@listes.uclouvain.be is available for discussions.

A few IPv6 routing examples using IPMininet designed to help students using the CNP3 ebook [5], are available at https://github.com/cnp3/RoutingExamples.

A list of iBGP labs [18], supporting CNP3 ebook, is available at https://github.com/cnp3/IPMininet-iBGP.

REFERENCES

[1] FRRouting. https://github.com/FRRouting/frr, Accessed Jun-20-2020.

[2] Ahrenholz, J., Danilov, C., Henderson, T. R., and Kim, J. H. Core: A real-time network emulator. In *MILCOM 2008-2008 IEEE Military Communications Conference* (2008), IEEE, pp. 1–7.

[3] Beckett, R., Mahajan, R., Millstein, T., Padhye, J., and Walker, D. Don't mind the gap: Bridging network-wide objectives and device-level configurations. In *Proceedings of the 2016 ACM SIGCOMM Conference* (2016), pp. 328–341.

[4] Bonaventure, O., Duchêne, Q. D. C. F., Gego, A., Jadin, M., Michel, F., Piraux, M., Poncin, C., and Tilmans, O. Open educational resources for computer networking. *ACM SIGCOMM Computer Communication Review* (July 2020).

[5] Bonaventure, O., et al. Computer networking : Principles, protocols and practice, 2019. https://www.computer-networking.info - Accessed Jun-20-2020.

[6] Cisco. Cisco learning labs. https://learningnetworkstore.cisco.com/cisco-learning-labs; Accessed Jul-6-2020.

[7] El-Hassany, A., Tsankov, P., Vanbever, L., and Vechev, M. Network-wide configuration synthesis. In *International Conference on Computer Aided Verification* (2017), Springer, pp. 261–281.

[8] Filsfils, C., Dukes, D., Previdi, S., Leddy, J., Matsushima, S., and Voyer, D. Ipv6 segment routing header (srh). RFC 8754, RFC Editor, March 2020.

[9] Grossman, J., et al. Graphical network simulator-3 (gns3). https://www.gns3.com/; Accessed Jul-6-2020.

[10] Group, V., et al. Ucb/lbnl/vint network simulator ns-2.

[11] Hasan, S., et al. Open/R: Open routing for modern networks. https://engineering.fb.com/connectivity/open-r-open-routing-for-modern-networks/, Accessed Jun-20-2020.

[12] Henderson, T. R., Lacage, M., Riley, G. F., Dowell, C., and Kopena, J. Network simulations with the ns-3 simulator. *SIGCOMM demonstration 14*, 14 (2008), 527.

[13] Holterbach, T., Bü, T., Rellstab, T., and Vanbever, L. An open platform to teach how the internet practically works. *ACM SIGCOMM Computer Communication Review 50*, 2 (2020), 45–52.

[14] Juniper Networks. Juniper networks virtual lab services. https://jlabs.juniper.net/home/; Accessed Jul-6-2020.

[15] Knight, S., Jaboldinov, A., Maennel, O., Phillips, I., and Roughan, M. Autonetkit: simplifying large scale, open-source network experimentation. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication* (2012), pp. 97–98.

[16] Lantz, B., Heller, B., and McKeown, N. A network in a laptop: rapid prototyping for software-defined networks. In *Hotnets 2010* (2010).

[17] Lebrun, D., and Bonaventure, O. Implementing ipv6 segment routing in the linux kernel. In *Proceedings of the Applied Networking Research Workshop* (2017), pp. 35–41.

[18] Mawait, M., and Rime, A. iBGP examples with IPMininet. https://github.com/cnp3/IPMininet-iBGP, Accessed Jun-20-2020.

[19] Narain, S., et al. Network configuration management via model finding. In *LISA* (2005), vol. 5, pp. 15–15.

[20] Narain, S., Levin, G., Malik, S., and Kaul, V. Declarative infrastructure configuration synthesis and debugging. *Journal of Network and Systems Management 16*, 3 (2008), 235–258.

[21] Nokia. My sr lab. https://networks.nokia.com/src/mysrlab; Accessed Jul-6-2020.

[22] Obstfeld, J., Knight, S., Kern, E., Wang, Q. S., Bryan, T., and Bourque, D. Virl: the virtual internet routing lab. In *Proceedings of the 2014 ACM conference on SIGCOMM* (2014), pp. 577–578.

[23] Pizzonia, M., and Rimondini, M. Netkit: network emulation for education. *Software: Practice and Experience 46*, 2 (2016), 133–165.

[24] Schlinker, B. Mininext | mininet extended. https://mininext.uscnsl.net/; Accessed Jul-6-2020.

[25] Tilmans, O., and Jadin, M. IPMininet. https://github.com/cnp3/ipmininet, Accessed Jun-20-2020.

[26] Varga, A. Omnet++. In *Modeling and tools for network simulation*. Springer, 2010, pp. 35–59.