Network Emulation using Linux Network Namespaces

Daniel Schubert, Benedikt Jaeger*, Max Helm*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: ga59tek@mytum.de, jaeger@net.in.tum.de, helm@net.in.tum.de

Abstract—**Testing the behaviour of computer networks can be done with specialized testbeds but they can be expensive and are hard to reconfigure. Therefore other methods like network emulation are used. In an implementation of an emulator, virtual machines can serve as network nodes. However, a more lightweight approach is based on Linux network namespaces. In this paper we describe fundamental emulation features of the Linux operating system that are useful in network emulation. Furthermore, we present Mininet, an emulator that harnesses those features. We examine the API of Mininet and show what happens in the background on a lower level. Finally we present data on the performance of Mininet.**

Index Terms—**network emulation, virtualization, software-defined networks, OpenFlow**

1. Introduction

Computer networks are already quite complex systems when they contain only a few nodes, with larger network topologies their behaviour gets even more unpredictable and so there is a need for testing. One option is to make use of a testbed consisting of several machines connected to each other but this approach is expensive, inflexible and does not scale very well. Therefore, simulation and emulation frameworks are useful especially in situations where the effects of introducing changes to the network have to be evaluated dynamically. These changes could relate to a protocol, the network architecture or the address scheme. In software defined networks where the functionality of the network can in principle evolve very quickly, such a platform for rapid prototyping can be helpful. Various prototyping environments are already available. In many of those, virtual machines represent nodes which then are connected into a network by virtual interfaces. However there is a different approach that was chosen for the open source network emulator Mininet. There the network is built from processes running in separate Linux network namespaces which are connected by pairs of virtual Ethernet devices. This offers a more lightweight way of network emulation. The remainder of this paper is organized as follows. Section 2 gives a detailed description of network namespaces and other emulation features of the Linux operating system. The Mininet network emulation platform is presented in Section 3, focussing on its API and inner workings. In Section 4 other simulation and emulation tools are discussed. Finally, a conclusion is given in Section 5.

2. Linux virtualization features

In this section we explain the virtualization features of the Linux operating system that are used for the implementation of Mininet.

2.1. Linux Network Namespaces

The concept of namespaces comes in different varieties in the Linux operating system. The common purpose is to offer spaces where processes can be executed in isolation of others regarding various system resources. One of those varieties are network namespaces. They provide processes or groups of processes with their individual network stack including routing tables, network devices, ports and firewall rules among other things. A new network namespace can be created in different ways using the namespace API. One way is to use the clone system call which creates a new process. If the `CLONE_NEWNET` flag is given as an argument the process is provided with a new namespace. Another way is to use the unshare system call from a child process again with the `CLONE_NEWNET` flag to separate it from its parent. At the time of their creation namespaces just contain a private loopback device. Physical network devices can be moved between namespaces but they can always only belong exactly to one of them [1] [2].

2.2. Virtual ethernet devices

In order to enable communication between different network namespaces there exist virtual ethernet devices (veth). They come as pairs and can be thought of as a pipe connecting two namespaces. Using the command shown in Listing 1 a pair of veth interfaces named <name1> and <name2> can be created and the latter is put in the network namespace <netns>. As a result, a connection between the root namespace and the namespace <netns> is established [3].

Listing 1: Shell command to create a virtual ethernet device pair

```
ip link add name <name1> type veth
peer name <name2> netns <netns>
```

2.3. Control groups

Control groups (cgroups) are to some extent similar to namespaces because they form an environment for processes with a modified view on system resources. Their purpose is to track and limit the access of processes to resources like cpu time, memory and devices or restrict the number of processes that can be created. Control groups are a hierarchical structure implemented as a pseudo-file-system. To create a new cgroup a folder is added to that file-system. Processes can be assigned to a cgroup by adding their process id to the group's cgroup.procs file. A process can only be part of one group and is automatically removed from any other group on its reassignment. The actual limitation of the resources is carried out by kernel components called controllers or subsystems which are mounted on the file-system. The limits of a cgroup are defined by values written in attribute files of a cgroup folder. [4]

2.4. Traffic control

Apart from controlling the environment a process is running in, it is also possible to influence the network traffic between network namespaces. This is done by influencing the handling of packets at the interfaces of namespaces using Linux traffic control (tc). This way for example the bandwidth of a link can be decreased [5].

3. Mininet

Mininet is an emulator that aims at providing a platform for rapid prototyping of large software defined networks consisting of hundreds of nodes. By using virtualization features on the operating system level it is very lightweight and can therefore be run on commodity hardware. It can be used interactively through a command line interface but there also exists a Python API that allows for the creation of complex network structures by small scripts. In fact almost the complete project itself is written in Python with only some time critical parts implemented in C [6].

3.1. Components

The components emulated by Mininet are hosts, switches, controllers and links. Mininet hosts are simply shell processes that have been given their own network namespaces. Software OpenFlow switches take over the tasks of hardware switches in real networks. By default they run in the root namespace. Typically, controllers are the parts that are tested and therefore beside using emulated ones, it is possible to connect real controllers to the virtualized network. This only requires IP-level connectivity between the controllers and the emulated switches. In order to connect the different nodes of the network virtual links are used. Each link consists of a virtual Ethernet device pair that acts like a tunnel between two virtual interfaces of two different network namespaces [6].

3.2. API

The Miniet API is divided into three levels of abstraction. The low-level API which comprises the base classes for the nodes and links that make up the network, the mid-level API that offers methods that help with the construction of a network as well as with its configuration and eventually the high-level API that provides a class representing a reusable network topology that can be parametrized and instantiated using the command line interface [7].

3.2.1. Low-level API. The most interesting things for us happen at the low-level API because there we can observe the utilization of the operating system's virtualization features. In Listing 2 you can see how the most basic network, containing only two hosts, a connecting switch and a controller can be constructed. The resulting network topology can be seen in Figure 1.

Listing 2: Construction of a simple network using Mininet's low-level API. (Modified from [7])

```
 1  h1 = Host('h1')
 2  h2 = Host('h2')
 3  s1 = OVSSwitch('s1', inNamespace=
        False)
 4  c0 = Controller('c0', inNamespace=
        False)
 5  Link(h1, s1)
 6  Link(h2, s1)
 7  h1.setIP('10.1/8')
 8  h2.setIP('10.2/8')
 9  c0.start()
10  s1.start([ c0 ])
11  print h1.cmd('ping -c1', h2.IP())
```

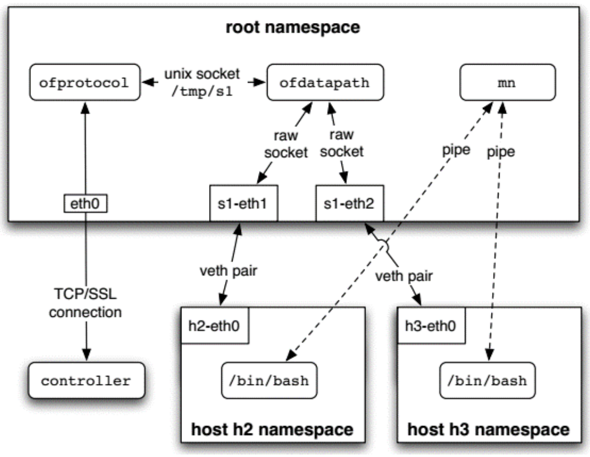

Figure 1: Simple network created by Mininet [6].

At this level, the base classes for the network's components are used directly. In Figure 2 you can see that the classes for three of the four main components of the network e.g. hosts, switches and controllers share a common parent class which is an abstraction for a network node. In the class constructor a new process is spawned that calls a subprogram written in C to create its own new network namespace by using the unshare system call with the `CLONE_NEWNET` flag. The basic Host class which is used in line 1 and 2 in Listing 2 does not differ from its superclass. There are neither new fields nor any new methods. This is different to the CPULimitedHost class where the id of the process started during the instantiation is moved to a cgroup file in order to control the CPU time

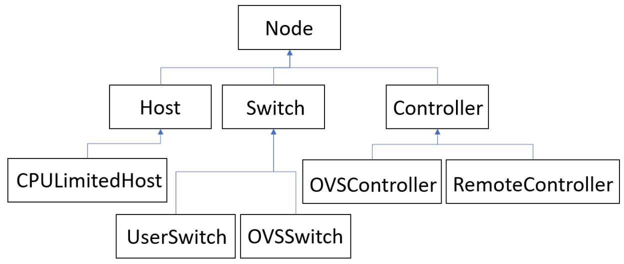

Figure 2: Class diagramm of a part of the Node class hierarchy.

allocated to it. In line 3 an open vSwitch is instantiated that eventually should bridge the two hosts and in line 4 an instance of a controller is created to determine the behaviour of the switch. In order to connect the switch to the hosts two Link objects are used. The basic Link object is just a pair of virtual Ethernet devices which is created as shown in Listing 1. In line 7 and 8 the IP adresses of the default interfaces of the hosts are set to 10.0.0.1 and 10.0.0.2 respectively. The default interface is the one with the lowest port number. In this case it is equal to the virtual Ethernet interfaces that were set up before because the network namespaces of the hosts are created empty except for a loopback interface that is not taken into consideration here. After that the controller's start method is called which brings up an OpenFlow reference controller that listens on port 6653 of the root network namespace. In line 10 an open vSwitch is started using the command shown in Listing 3.

Listing 3: Shell command to start an open vSwitch

```
1  ovs-vsctl add-br <name>
2  -- set bridge <name>
3  controller=[<controllerIds>]
4  -- add-port <name> <intf>
```

Line 1 in Listing 3 shows the basic command to add a bridge called <name>. In line 2-3 the controllers responsible for the switch's behaviour are set and in line 4 an interface is added to the bridge. Line 4 is executed two times in this example adding the two different interfaces that have been created during the instantiation of the two links between the switch and the hosts. Finally, in line 11 of Listing 2 host h2 is pinged from host h1. The cmd method of the Node superclass takes a list of arguments that are combined to a string and run in the shell of the respective node [8].

3.2.2. Mid-level API. If we want to build the same simple network using the mid-level API we can use the code that is shown in Listing 4.

Listing 4: Command to start an open vSwitch

```
1  net = Mininet()
2  h1 = net.addHost('h1')
3  h2 = net.addHost('h2')
4  s1 = net.addSwitch('s1')
5  c0 = net.addController('c0')
6  net.addLink(h1, s1)
7  net.addLink(h2, s1)
8  net.start()
9  print h1.cmd( 'ping␣-c1', h2.IP())
10 net.stop()
```

When we compare it to the code written using the low-level API we can see some differences. First of all there is a Mininet object which is an abstraction of the network. The network can be started and stopped as a whole as done in line 8 and 10 respectively. Therefore, it is not necessary to start switches or controllers separately. Furthermore, there is no need to manually set IP addresses of interfaces.

3.2.3. High-level API. In the high-level API there is a Topo class that represents a reusable network topology that can be parametrized. This class contains a build method that can be overwritten which orchestrates the creation of a network in essentially the same way as in the mid-level API. A Topo object can be handed to the Mininet constructor as a parameter or can be used as an argument for the command line interface.

3.3. Evaluation of Performance and Accuracy

The inventors of Mininet themselves published a report where they compared the bandwidths that were measured in small networks in the emulator with those of equivalent topologies on a testbed with eight machines. They observed similar TCP results but the results of the emulator were more repeatable and consistent [9].

In a larger study Isaia and Guan [10] examined Mininet with regard to nine different performance categories, including setup time, teardown time, CPU usage, CPU cores load balancing, RAM usage, initial ping delay (IPD), average ping delay, no response failure rate and fair share of resources. Setup and teardown time describe how long it takes to create and destruct a given network topology. To evaluate the CPU usage they took two different measures: the initial CPU usage which is the CPU load after creating the network but before starting to send data across it and the CPU usage during experimentation which is the average CPU load during a specified experiment. For measuring the CPU cores load balancing, an experiment is divided into time intervals. For each of them the standard deviation of core usage is calculated and finally the average is taken. It is a measure of how well the CPU load can be distributed to different cores which is important for scalability. The initial ping delay is the time it takes to ping a node at the start of an experiment. It is significantly larger than at later time points because at the beginning the OpenFlow switches do not contain the necessary flow table rules which first have to be added by a controller. Therefore, in the average ping delay measure the IPD is excluded. No response failure rate is the percentage of unsuccessful ping commands. Fair share of resources in that study was calculated as the coefficient of variation of ping delay between all the hosts when performing a ping command simultaneously. Five different setups with their own network topologies and communication patterns have been tested in 4 different network sizes and each experiment was done 30 times on two different systems that differed in the number of cores, the amount of RAM and the size of the hard disk. The setups covered various bottlenecks in network communication. The data generated from the experiments showed that the setup time of a network is strongly influenced by the number of switches. In a network of 1000 hosts and one swith the setup time was under ten seconds whereas in a network comprised of two hosts and 1000 switches it reached almost four minutes. But setup time did not always increase linearly with the size of the network and there was no benefit from using the more powerful system. The authors of the study also stated that CPU usage is generally good and that load balancing worked well being positively effected by the number of switches. The initial ping delay is very large compared to the average ping delay and it grows as the number of nodes in the network increases. This is natural because when a connection is used for the first time the forwarding rules have to be added to the switch by a controller. The no response failure rate played a role in larger networks or when the path of a ping packet was long. The fair share of resources measure also gets worse with an increasing number of network nodes. [10]

4. Related work

Apart from Mininet there exist other tools to simulate or emulate computer networks. Simulators try to mimic the behavior of a system model in a more abstract way whereas with emulators the same code can be executed as in the real system. A disadvantage of emulators may be that they run slower than the actual hardware and therefore are not always able to reproduce a realistic timing. In simulators the execution is more flexible and can even be faster then in the real system [11]. On the side of the simulation tools, there is ns-3, an open source discrete-event simulator that was mainly created to support education and research [12]. Others are fs-sdn [13], which as Mininet is targeted at SDN prototyping and the commercial EstiNet X Simulator [14]. An example for network emulation is Mahimahi, a record-and-replay tool that can be used for recording traffic from HTTP-based applications. Later Mahimahi can replay the traffic emulating the network structure that produced it. Linux network spaces are used here as well [15].

Although hundreds of nodes can be emulated on a single machine with Mininet at some point the resources come to an end where it is not possible to make the network larger. To handle this problem Blankstein et al. developed a distributed version of Mininet where the topology of a virtual network is split between multiple computers automatically [16]. This approach has now also been followed by the Mininet community and there is a cluster edition prototype available in the repositiory.

5. Conclusion

In this paper we explained fundamental emulation features of the Linux operating system that are useful in network emulation including network namespaces, virtual Ethernet devices, control groups and traffic control. Furthermore we presented the Mininet emulator, which is based on those features, and described the implementation of its main components. We examined Mininet's API that is divided into three abstraction levels and inspected how it interacts with the operating system to create a simple network. Finally we presented data on performance and accuracy of Mininet and mentioned other tools for network simulation and emulation.

References

[1] "namespaces(7) - linux manual page," http://man7.org/linux/man-pages/man7/namespaces.7.html, [Online; accessed 2019-06-11].

[2] network_namespaces(7) - linux manual page. [Online]. Available: http://man7.org/linux/man-pages/man7/network_namespaces.7.html

[3] veth(4) - linux manual page. [Online]. Available: http://man7.org/linux/man-pages/man4/veth.4.html

[4] "cgroups(7) - linux manual page," http://man7.org/linux/man-pages/man7/cgroups.7.html, [Online; accessed 2019-06-13].

[5] tc(8) - linux man page. [Online]. Available: https://linux.die.net/man/8/tc

[6] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org.eaccess.ub.tum.de/10.1145/1868447.1868466

[7] mininet. Introduction to mininet. [Online]. Available: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#api

[8] ——. Mininet sourcecode. [Online]. Available: https://github.com/mininet/mininet

[9] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Mininet performance fidelity benchmarks," 2012.

[10] P. Isaia and L. Guan, "Performance benchmarking of sdn experimental platforms," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, June 2016, pp. 116–120.

[11] C. Seifert, S. Reißmann, S. Rieger, and C. Pape, "Evaluation von virl, gns3 und mininet als virtual network testbeds in der hochschullehre," in *11. DFN-Forum Kommunikationstechnologien*, P. Müller, B. Neumair, H. Reiser, and G. Dreo Rodosek, Eds. Bonn: Gesellschaft für Informatik e.V., 2018, pp. 103–112.

[12] T. Henderson and M. Lacage. Network simulator 3. [Online]. Available: https://www.nsnam.org

[13] M. Gupta, J. Sommers, and P. Barford, "Fast, accurate simulation for sdn prototyping," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 31–36. [Online]. Available: http://doi.acm.org.eaccess.ub.tum.de/10.1145/2491185.2491202

[14] EstiNet. Estinet. [Online]. Available: https://www.estinet.com/ns/

[15] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for HTTP," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. Santa Clara, CA: USENIX Association, 2015, pp. 417–429. [Online]. Available: https://www.usenix.org/conference/atc15/technical-session/presentation/netravali

[16] A. Blankstein, S. A. Erickson, and M. Melara, "Mininet clustering," 2013.