Investigating Linux Network Behaviour Using Open-Source Network Emulators

Brian Linkletter

Open-Source Routing and Network Simulation
Ottawa, Canada
mail@brianlinkletter.ca

Abstract

There are several open source, GUI based, network emulation tools that are based on Linux containers or on virtual machines that could be used to replicate network scenarios including failures. This paper provides examples of trouble-shooting network problems created in emulated networks using Linux networking functions and introduces a sample of useful Linux network emulation tools.

Keywords

Network Emulation; Network Simulation; CORE; VNX, GNS3; UNetlab; Mininet; Cloonix; Open Source.

Introduction

To acquire hands-on knowledge of Linux networking operations , it is desirable to use a tool that can build complex virtual networks on a single, modestly-powered laptop computer. In this paper, we discuss several very useful projects, each of which is suitable for a different use-case.

Up-to-date information about open-source network emulation tools is not readily available. A few papers have been written that survey available tools but, since they were written years ago, they do not cover new tools and, in some cases, also cover projects that have been abandoned.

In this paper, I will provide a description of how to use a popular Linux-based network emulation tool, the CORE Network Emulator, to create network troubleshooting scenarios that would allow users to test Linux network operation procedures and that could potentially allow developers to test Linux networking software in both an ad-hoc manner and as part of a standardized virtual networking test bed. I will also provide a survey of some of the functional and well-supported open-source network emulation tools for Linux networking experiments.

Demonstrating troubleshooting

Open-source network emulators enable developers and users to rapidly create and emulate ad-hoc or scripted network topologies based on nodes running real software. In Figure 1, I show a screenshot of a network created using the Common open research Emulator (CORE) with seven routers and thirteen hosts and servers.

First, we may investigate the behavior of TCP in various network conditions. We may show theoretical throughput when link delays are zero and we may show more realistic throughput when we configure delay on network links. Next we may introduce link errors and use *Wireshark* to view how TCP handles missing packets. We may also investigate how routing protocols respond to changes in the network topology.

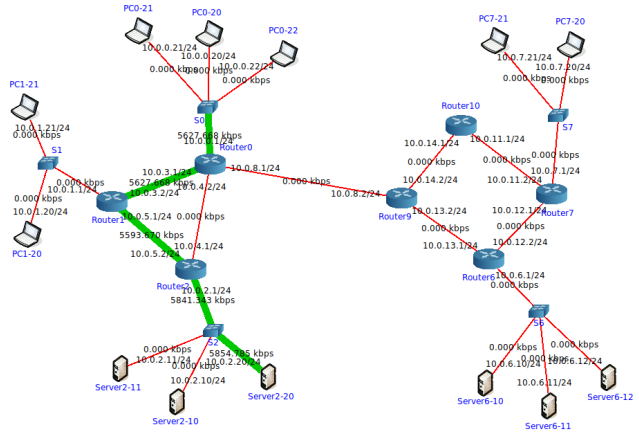

Figure 1. Network topology map in CORE network emulator, showing TCP traffic flowing between two nodes.

For example, we may run *iperf3* to create a TCP flow between two nodes in the emulated network: *PC0-21* and *Server2-20*.

With no delay configured in the emulated network, the round trip delay across the virtual network is less than 0.1 ms. So we expect to see very high throughput with any given TCP window size.

We first start an iperf3 server on Server2-20:

```
root@Server2-20:/tmp/pycore.43763/Server2-20.conf# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Then we run iperf3 on PC0-21 with a window size of 64K (some output deleted):

```
<7/PC0-21.conf# iperf3 -c 10.0.2.20 -O 2 -t 5 -w 64K
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-5.00   sec  10.6 GBytes  18.2 Gbits/sec    0  sender
[  4]   0.00-5.00   sec  10.7 GBytes  18.3 Gbits/sec     receiver
```

Next, we configure the links in the network with delay adding up to 50 ms round trip time. When we run iperf3 again, we see significantly reduced TCP throughput, as expected:

```
<97/PC0-21.conf# iperf3 -c 10.0.2.20 -O 2 -t 5 -w 64K
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-5.00   sec  5.98 MBytes  10.0 Mbits/sec    4  sender
[  4]   0.00-5.00   sec  6.04 MBytes  10.1 Mbits/sec     receiver
```

We may perform more experiments where we change the delay on multiple links in the network and observe the affect on TCP throughput.

As another example, we may emulate a faulty link in the network by configuring bit errors on a link in the network. We may observe TCP messages using Wireshark and monitor TCP retransmissions. We will configure one of the links with a 30% error rate. This will cause multiple missing packets.

The first indication of trouble would be greatly reduced TCP throughput. Testing with iperf shows the retransmissions:

```
<7/PC0-21.conf# iperf3 -c 10.0.2.20 -O 2 -t 5 -w 64K
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-5.00   sec  93.3 KBytes   153 Kbits/sec   19   sender
[  4]   0.00-5.00   sec  84.8 KBytes   139 Kbits/sec      receiver
```

We observe in Figure 2 a Wireshark packet capture that shows how TCP handles the missing packets by requesting retransmissions

As a final example, we may observe the affects that changes in the network topology have in the network. In this example, if we configure OSPF in the network, we may observe changes caused by changing link costs or by causing a fault in the network by disabling OSPF on one of the network links. We may observe OSPF updates using

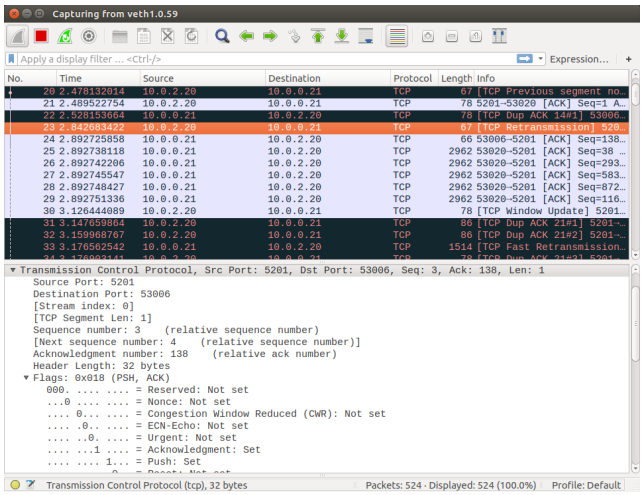

Figure 2. Wireshark packet capture showing retransmissions due to missing packets.

Wireshark or observe the effects by querying the OSPF database on different nodes in the network.

For example, in the network shown in Figure 1, we may observe that the OSPF database on *Router2* is as follows:

```
Router2# show ip ospf database

       OSPF Router with ID (10.100.0.3)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.100.0.1      10.100.0.1       206 0x8000000e 0xea43 5
10.100.0.2      10.100.0.2       811 0x8000000a 0x8fc8 4
10.100.0.3      10.100.0.3       815 0x80000009 0xf260 4
10.100.0.6      10.100.0.6       679 0x80000009 0xf234 4
10.100.0.7      10.100.0.7       680 0x80000009 0x83a4 4
10.100.0.9      10.100.0.9       717 0x8000000a 0xfb05 4
10.100.0.10     10.100.0.10      697 0x80000007 0x1533 3

                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
10.0.3.2        10.100.0.2       835 0x80000001 0xf208
10.0.4.2        10.100.0.1       854 0x80000001 0xf306
10.0.5.2        10.100.0.3       815 0x80000001 0xee07
10.0.11.1       10.100.0.10      697 0x80000001 0x15c8
10.0.12.1       10.100.0.7       680 0x80000001 0xeff3
10.0.13.2       10.100.0.9       717 0x80000001 0xeeee
10.0.14.2       10.100.0.9       757 0x80000001 0x0ccc
```

We may then introduce a change in the network by, for example, disabling OSPF on *Router9*. This has the effect of removing visibility of half the network. We may observe the OSPF LS update in Wireshark, as shown in Figure 3, and we may observe the change in the OSPF database on *Router2* as follows:

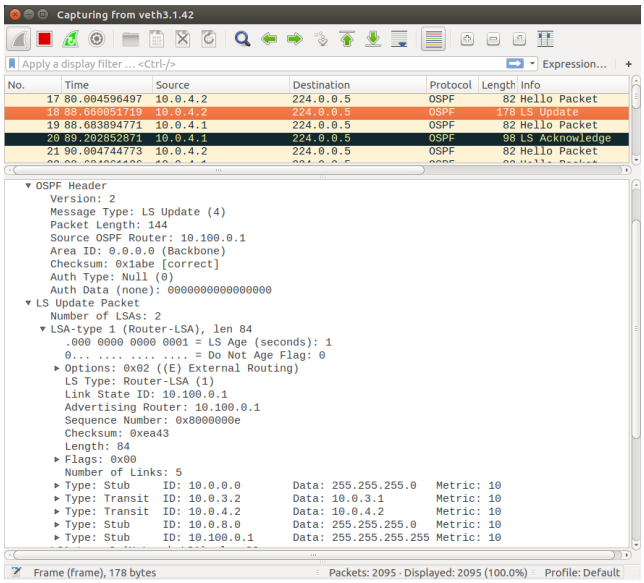

Figure 3. Wireshark packet capture showing OSPF update after disabling OSPF on *Router9*

```
Router2# show ip ospf database

       OSPF Router with ID (10.100.0.3)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.100.0.1      10.100.0.1      1026 0x80000010 0xe645 5
10.100.0.2      10.100.0.2      1596 0x8000000c 0x8bca 4
10.100.0.3      10.100.0.3      1547 0x8000000b 0xee62 4


                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
10.0.3.2        10.100.0.2       856 0x80000003 0xee0a
10.0.4.2        10.100.0.1       706 0x80000004 0xed09
10.0.5.2        10.100.0.3      1697 0x80000003 0xea09
```

Performance

When using network emulators, we must be aware that performance may not match real-world performance where resources on the host computer are limited. We must be careful to create scenarios that do not exceed available memory or processing power or our results will vary greatly. However, by choosing emulators that use light-weight virtualization technology such as Linux containers, by limiting the number of virtual machines running on the host computer, or by using a more powerful host computer, it is possible to reproduce results using network emulators. [5]

Open-Source Network Emulators

Below is a list of open-source network emulators discussed in this paper that run on Linux and support Linux-based virtual nodes, either containers or virtual machines.

- Cloonix
- Common Open Research Emulator (CORE)
- EVE (and UNetLab)
- GNS3
- IMUNES
- Mininet
- Netkit
- VNX

Cloonix

The Cloonix Network Emulator provides a simple and easy-to-use graphical user interface. Cloonix uses QEMU/KVM to create virtual machines. Cloonix provides a wide variety of pre-built file systems that can be used as virtual machines and provides simple instructions for creating other virtual machine root file systems. Cloonix has an active development team, who update the tool every two or three months and who are very responsive to user input.

Users and developers who prefer to use Linux shell scripts will be comfortable working with the text-based Cloonix topology files. Cloonix provide a command-line interface that can be scripted using standard shell scripts. Cloonix will export a network scenario as a shell script. Users may edit the shell scripts to create fully-configured setup scripts that, when started, will build and configure complex Linux networks. Since Cloonix uses virtual machines, it is appropriate for testing custom kernels and other Linux router distributions.

Common Open Research Emulator

The Common Open Research Emulator (CORE) provides a GUI interface and uses the network namespaces functionality in Linux. [1] This allows CORE to start up a large number of virtual machines quickly, because each node uses minimal resources. CORE supports the emulation of fixed and mobile networks.

CORE will run on Linux and on FreeBSD. It is written in Python and users may modify the program to support additional network services.

The CORE topology files are readable text and may be saved as a standard text file or in XML format. Users may edit the topology files to create fully-configured, complex network scenarios. Core services are implemented in Python so users may modify CORE services to create new ones.

Since CORE uses Linux namespaces to implement the nodes in the virtual network, users must ensure that they specify which parts of the file system need to be mounted in a mount namespace for each node.

EVE and UNetlab

EVE and UNetLab are network emulators that support virtualized commercial router images (such as Cisco and NOKIA) and open-source routers such as Linux. They use

Dynamips and IOS-on-Linux to support Cisco router and switch images, and KVM/QEMU to support all other devices. Each is available as a virtual machine image and may also be installed on a dedicated server running Ubuntu Linux.

EVE-NG is the next-generation of UNetlab. Both UNetLab and EVE-NG are open-source and post source code on GitHub for UNetLab and on GitLab for EVE-NG.

Users must perform some extra configuration steps to enable EVE to support fully functioning Linux systems.

GNS3

GNS3 is a graphical network simulator focused mostly on supporting Cisco and Juniper software. GNS3 has a large user base, made up mostly of people studying for Cisco exams, and there is a lot of information freely available on the web about using GNS3 to simulate Cisco equipment. However, comprehensive information about using GNS3 to support Linux nodes is only recently being developed, due to enthusiasm among network engineers for software-defined networking exercises.

GNS3 can also be used to simulate a network composed exclusively of VirtualBox and/or KVM/QEMU virtual machines running open-source software. GNS3 provides a variety of prepared open-source virtual appliances, and users can create their own.

IMUNES

The Integrated Multi-protocol Network Emulator/Simulator (IMUNES) runs on both the FreeBSD and Linux operating systems. It uses the kernel-level network stack virtualization technology provided by FreeBSD. It uses Docker containers and Open vSwitch on Linux. [2]

IMUNES supports a graphical user interface. It works well and offers good performance, even when running IMUNES in a VirtualBox virtual machine.

Mininet

Mininet is designed to support research in Software Defined Networking (SDN) technologies. It uses Linux network namespaces as its virtualization technology to create virtual nodes. [3] The web site indicates that the tool can support thousands of virtual nodes on a single operating system. Mininet is most useful to researchers who are building SDN controllers and need a tool to verify the behavior and performance of SDN controllers. Knowledge of the Python scripting language is very useful when using Mininet.

The Mininet project provides excellent documentation and, judging from the activity on the Mininet mailing list, the project is actively used by a large community of researchers.

Some researchers have created forks of Mininet that focus on specific technologies, in addition to OpenFlow-based SDN. other projects based on Mininet are:

- Mini-NDN
- Mini-CCNx
- Mininet-WiFi
- ESCAPE

VNX

VNX supports two different virtualization techniques: LXC containers and KVM virtual machines. It also supports FreeBSD virtual machines. [6] It uses an XML-style scripting language to define the virtual network. It also supports chaining multiple physical workstations together to support distributed virtual labs that operate across multiple physical workstations. It is supported by a small but active community.

To create a VNX scenario, users edit a topology file in a specific format. This makes VNX harder to get started with than other network emulators that provide a GUI.

The VNX project provides a generous repository of pre-packaged networking scenarios. These scenarios make VNX attractive as a teaching tool.

Netkit

Netkit uses User Mode Linux as a virtualization technology. In a similar fashion as VNX, above, Netkit users must build a set of lab description files in located in directories related to the topology and using a custom network description language. [4] The lab description files may run Netkit commands that configure and control with the virtual nodes in the network topology.

Netkit also provides a generous repository of pre-packaged networking scenarios and also documents each scenario in a way that is meant to support an educational environment.

Other Network Emulators

Below is a list of more open-source network emulator projects that are not described in this paper but are still active projects worth investigating. They are either relatively new, based on older visualization technology, or are dedicated to a special purpose.

- OFNet
- Shadow
- NetMirage
- Yet Another Network Simulator
- Marionnet

Conclusion

I discussed how open-source network emulators may be used to experiment with failure scenarios in a virtual network. I also briefly reviewed the open-source network emulators currently available.

References

1. Comparison of CORE Network Emulation Platforms, Proceedings of IEEE MILCOM Conference, 2010, pp.864-869.
2. A network testbed for commercial telecommunications product testing
D. Salopek, V. Vasic, M. Zec, M. Mikuc, M. Vasarevic, V. Koncar; in Proceedings of the Softcom 2014 22th International Conference on Sotware, Telecommunications and Computer Networks, Split, September 2014.
3. Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A network in a laptop: rapid prototyping for software-defined networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX). ACM, New York, NY, USA, , Article 19
4. Maurizio Pizzonia and Massimo Rimondini. 2008. Netkit: easy emulation of complex networks on inexpensive hardware. In Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities (TridentCom '08). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, , Article 7
5. Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. 2012. Reproducible network experiments using container-based emulation. In Proceedings of the 8th international conference on Emerging networking experiments and technologies (CoNEXT '12). ACM, New York, NY, USA, 253-264.
6. D. Fernández, F. J. Ruiz, L. Bellido, E. Pastor, O. Walid and V. Mateos, Enhancing Learning Experience in Computer Networking through a Virtualization-Based Laboratory Model, International Journal of Engineering Education Vol. 32, No. 6, pp. 2569–2584, 2016.

Bibliography

IMUNES web site
https://github.com/imunes/imunes

Cloonix web site
http://virtual-network-kvm.net/

Mininet web site
http://mininet.org/

Mininet-wifi web site
http://www.ramonfontes.com/mininet-wifi/

VNX web site
http://www.dit.upm.es/vnx

Netkit web site
http://wiki.netkit.org

GNS3 Commnity
https://www.gns3.com/

EVE web site
http://www.eve-ng.net/

OFNet web site
http://sdninsights.org/

Shadow network emulator wen site
https://shadow.github.io/

Marionnet web site
http://www.marionnet.org/site/index.php/en/

Netmirage web site
https://crysp.uwaterloo.ca/software/netmirage/

Yet Another Network Simulator web site
https://github.com/kennethjiang/YANS

Mini-NDN web site
https://github.com/named-data/mini-ndn

Mini-CCNx web site
https://github.com/chesteve/mn-ccnx/wiki

Mininet-WiFi web site
https://github.com/intrig-unicamp/mininet-wifi

ESCAPE web site
http://sb.tmit.bme.hu/mediawiki/index.php/ESCAPE

Open-Source Routing and Network Simulation blog
http://www.brianlinkletter.com

Author Biography

Brian Linkletter is a network professional with 25 years experience. He writes a blog about open-source networking and network simulation. He currently works for a large telecommunications equipment vendor and is based in Ottawa, Canada.