

Emulating LAN/WAN Technologies from the Comfort of your Own Home with GNS-3

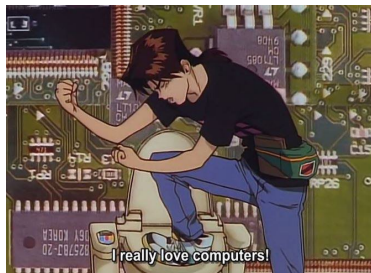
Wesley Coakley

E-Mail w@wesleycoakley.com

Homepage <http://wesl.ee>

Linux User Group @ N.C. State

October 20, 2020



Outline

- 1 Overview
 - What is Network Emulation?
 - Deploying Technology on an Emulated Network
- 2 A Survey of Existing Emulation Technologies
 - docker-topo
 - k8s-topo
 - Common Open Research Emulator
 - GNS-3
- 3 Using GNS-3
 - Generating a topology
 - Running an Actual Router
 - Turning on OSPFv2
- 4 Closing Remarks

Outline

- 1 Overview
 - What is Network Emulation?
 - Deploying Technology on an Emulated Network
- 2 A Survey of Existing Emulation Technologies
 - docker-topo
 - k8s-topo
 - Common Open Research Emulator
 - GNS-3
- 3 Using GNS-3
 - Generating a topology
 - Running an Actual Router
 - Turning on OSPFv2
- 4 Closing Remarks

What is Network Emulation?

It is the method of creating a virtual network testbed which is suitable for testing, benchmarking, and developing networked applications

- It is distinct from network *simulation* which only models the network and none of the applications in it
- ... also distinct from a “lab” which relies on physical hardware

Although we *can* use real hardware coupled with our emulator, I will only talk about only pure, virtual networks

Why Network Emulation? (WAN)

When developing a networked program, it is important to understand how it will behave on the Internet (WAN) at-scale:

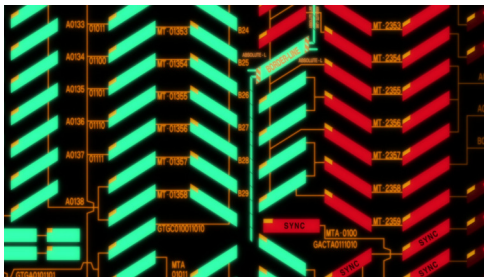
- Peer-to-peer Applications
 - Bitcoin
 - Torrent Swarms
 - Soulseek
- Routing Suites
 - Cisco IOS
 - FRR
 - GoBGP
- Overlay Networks
 - GNU Social
 - The Onion Router

Why Network Emulation? (LAN)

Even on LAN, we may be interested in simulating things like:

- Home networks with many wired devices
- Campus networks with many types of clients
 - Wired
 - Bluetooth Personal Area Networks
 - 2.4GHz / 5GHz Wireless
- Datacenters using Internal Gateway Protocols (e.g.)
 - Routing Information Protocol (RIP)
 - Open Shortest Path First (OSPF)
 - Internal Border Gateway Protocol (iBGP)

Instead of running these applications on the open Internet...



Let's just emulate the Internet (err, just a part of it)!

Pros

- Inexpensive yet scalable
- Able to tune network parameters easily

Cons

- Kinda complicated
- (Can be) resource-intensive

Outline

- 1 Overview
 - What is Network Emulation?
 - Deploying Technology on an Emulated Network
- 2 A Survey of Existing Emulation Technologies
 - `docker-topo`
 - `k8s-topo`
 - Common Open Research Emulator
 - GNS-3
- 3 Using GNS-3
 - Generating a topology
 - Running an Actual Router
 - Turning on OSPFv2
- 4 Closing Remarks

Approach / Design

What is our atomic “unit” of design?

- We *could* use full-blown VMs...
 - QEMU + KVM
 - Virtualbox
 - Something else...
- Or we could use a “lightweight” alternative
 - Containers!

Luckily there is already software which can network containers together! Most of these suites expect to use Docker so we'll be using that

Container Considerations

We'll be using containers for our emulation, however there are some things to consider before we get started:

- Limited to host-machine capabilities / kernel
 - No BSD
 - No Windows
 - No Items
 - Final Destination
- *Kernel panics will crash the host*



docker-topo

docker-topo will connect Docker containers for you very quickly, using either (1) `macvlan`, (2) `veth`, or (3) `bridge` drivers in a user-defined topology

```
links:
- endpoints: ["spine-1:eth0", "spine-2:eth0", "leaf-1:eth0", "leaf-2:eth0", "leaf-3:eth0"]
  driver: bridge
- endpoints: ["spine-1:eth1", "leaf-1:eth1"]
- endpoints: ["spine-1:eth2", "leaf-2:eth1"]
- endpoints: ["spine-1:eth3", "leaf-3:eth1"]
- endpoints: ["spine-2:eth1", "leaf-1:eth2"]
- endpoints: ["spine-2:eth2", "leaf-2:eth2"]
- endpoints: ["spine-2:eth3", "leaf-3:eth2"]

VERSION: 2
driver: veth
PREFIX: ospf
CUSTOM_IMAGE:
  spine: frr:centos-8-0b1dc328a9
  leaf: frr:centos-8-0b1dc328a9
CONF_DIR: ./config
PUBLISH_BASE: 9000
topo-old/frr/spine-leaf.yaml lines 1-18/18 (END)
```

Figure: A spine-leaf configuration in `docker-topo`

Thoughts on docker-topo

I like a lot of things about docker-topo

- Very simple interface
- Does one thing and does it well

Unfortunately docker-topo has a number of short-comings:

- Not actively developed
- Built mostly for Arista Containerized EOS (cEOS)
- No way to visualize topologies
- Connecting nodes can be tedious
- No simple way to assign static IPs

k8s-topo

k8s-topo is written by the author of docker-topo! It uses Kubernetes to build and simulate complex network topologies (similar to docker-topo)

- Actively developed
- Similar config. structure to docker-topo
- I've [heard it's pretty good](#)

While it's great for testing routing technologies (e.g. Cisco XRv, FRR, cEOS) this is not ideal for (e.g.) testing P2P networks, which is one of our end-goals

k8s-topo Network Visualization

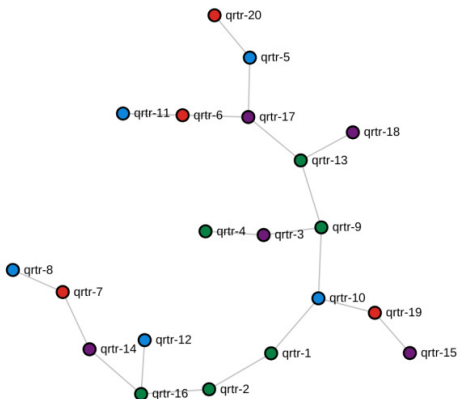
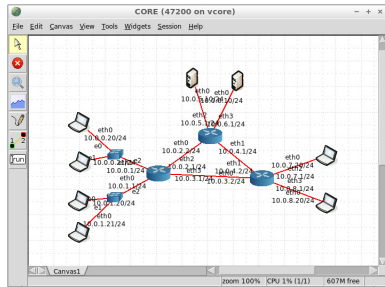


Figure: An example topology using k8s-topo

Common Open Research Emulator

C.O.R.E. is a mature program (developed since 2005) which emulates network connections between virtualized peers.

- Many services are supported out of the box
 - FRR
 - BIRD
 - XORP
- Can connect and interact with physical networks (v. neat)
- Supports multiple / distributed hosts



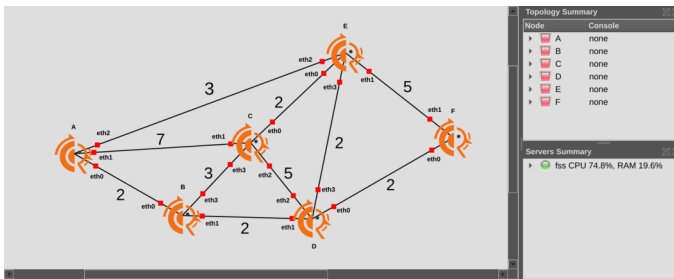
How C.O.R.E. stacks up (for our use-case)

C.O.R.E. satisfies almost everything we need to build our topology. Unfortunately, its Docker support is quite limited, and is driven entirely by a set of (relatively undocumented) scripts buried in the repository.

...

GNS-3

GNS-3 is a well-known network emulator that is popular among students studying for their CCNA



- Wireshark integration
- Docker support
- Many vendors are already supported
- Network visualization

GNS-3 and Docker Containers

GNS-3's Docker support allows you to

- Connect and network Docker containers in any topology
- Easily mount persistent volumes, useful for:
 - Software config files
 - Special networking parameters
 - Logs and automation
- Create containers on-the-fly

You can even interact with these containers using typical Docker tooling from the comfort of your own shell!

Outline

- 1 Overview
 - What is Network Emulation?
 - Deploying Technology on an Emulated Network
- 2 A Survey of Existing Emulation Technologies
 - docker-topo
 - k8s-topo
 - Common Open Research Emulator
 - GNS-3
- 3 Using GNS-3
 - Generating a topology
 - Running an Actual Router
 - Turning on OSPFv2
- 4 Closing Remarks

Generating a topology

Docker containers can be accessed from Edit → Preferences

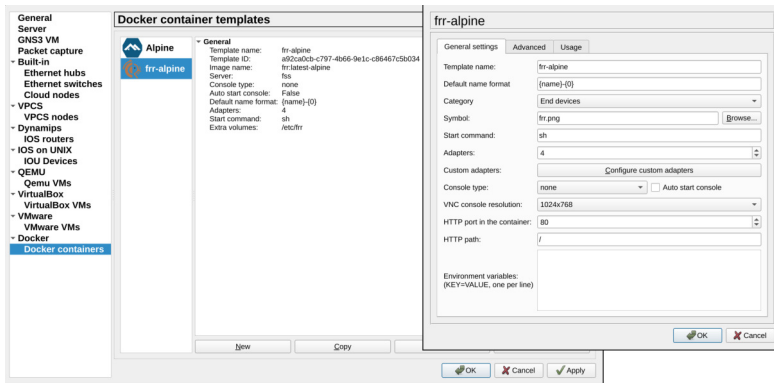


Figure: Importing an existing Docker container

Generating a topology (cont.)

With a container imported, you can connect two together using links attached to the container's network adapters

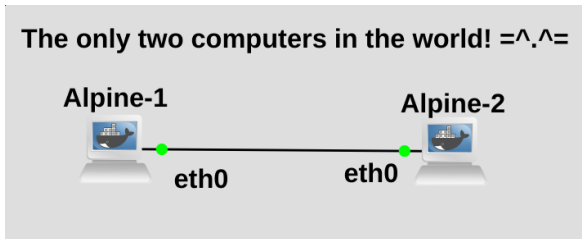


Figure: Two connected Docker containers in GNS-3

You can change link jitter, latency, loss, and corruption!

Generating a topology (cont.)

- 1 Remote into each container (or edit the volume)
- 2 Configure each /etc/network/interfaces
- 3 Reboot topology and confirm connectivity

Now the hosts can communicate with each-other as if they were on the same physical network!

```

ucoatley@fss:~$ docker exec -it 3b2bfcf44534 sh
/ # hostname
Alpine-1
/ # ip a show dev eth0
19: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNK
NDMKN qlen 1000
    link/ether 1e:e9:7b:e5:ac:59 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.100/24 scope global eth0
        valid_lft forever preferred_lft forever
    inet5 fe80::1ce9:7bff:fe5:ac59/64 scope link
        valid_lft forever preferred_lft forever
/ # ping -c 1 192.168.0.101
PING 192.168.0.101 (192.168.0.101): 56 data bytes
64 bytes from 192.168.0.101: seq=0 ttl=64 time=0.261 ms

--- 192.168.0.101 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.261/0.261/0.261 ms
/ # █

ucoatley@fss:~$ docker exec -it 7c13b7713c84 sh
/ # hostname
Alpine-2
/ # ip a show dev eth0
1b: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNK
NDMKN qlen 1000
    link/ether d6:a5:93:0a:27:60 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.101/24 scope global eth0
        valid_lft forever preferred_lft forever
    inet5 fe80::d4a5:93ff:fe0a:2760/64 scope link
        valid_lft forever preferred_lft forever
/ # █

```

Figure: Demonstrating connectivity between Alpine containers

Will it route?

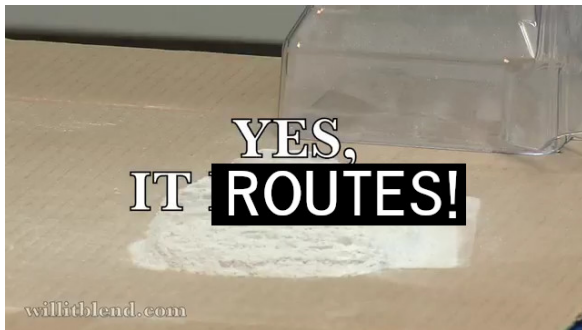
Alpine Linux is great, but can it route?



That is the question. . .

Yes! Alpine Linux plus FRRouting

Using a routing stack (we'll use FRRouting) we can build a Docker image for our Alpine Linux router!



Docker build instructions are [here](#).

What is FRRouting?

FRRouting (FRR) is free and open-source software that implements a number of Internet routing protocols

It is used to route traffic around:

- Datacenters
- Internet Service Providers
- Network labs
- ... and by students / hobbyists / tinkerers

We'll use FRR to inject routing information into our virtual GNS-3 network to study how the containers react to network stimulus

Building an FRR Docker container



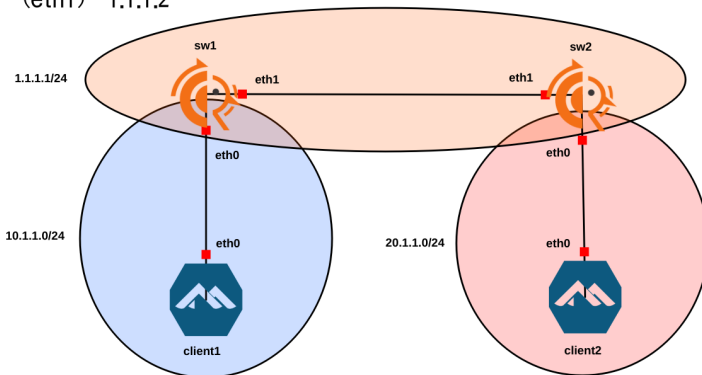
```
wcoakley@fss:~$ git clone 'https://github.com/FRRouting/frr' && cd frr
Cloning into 'frr'...
remote: Enumerating objects: 141003, done.
remote: Total 141003 (delta 0), reused 0 (delta 0), pack-reused 141003
Receiving objects: 100% (141003/141003), 77.90 MiB | 10.79 MiB/s, done.
Resolving deltas: 100% (111903/111903), done.
wcoakley@fss:~/frr$ which docker
/usr/bin/docker
wcoakley@fss:~/frr$ docker build -f docker/alpine/Dockerfile .
```

Figure: It's *just* that easy! (after installing Docker)

You could also build on top of Centos 8, Centos 7, Debian, or you could even roll your own container!

Building a topology

```
client1 (eth0) 10.1.1.2  
client2 (eth0) 20.1.1.2  
sw1 (eth0) 10.1.1.1  
sw2 (eth0) 1.1.1.1  
sw1 (eth1) 20.1.1.1  
sw2 (eth1) 1.1.1.2
```



Building a topology (cont.)

Some notes about the above topology:

- `client1` and `client2` are on different subnets
- Without any special configuration we cannot see one client from the other

Until we tell the *routers* to be *routers* they will not route traffic!

Time for some routing magic:

- We could either write static routes in `/etc/network/interfaces ...`
- Or we could use one of FRR's routing protocols!

We'll pick the latter (this is actually easier / scalable here)

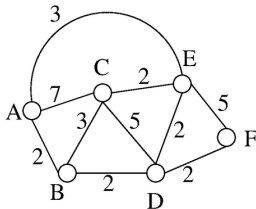
OSPFv2 Primer

FRR supports a routing protocol called “Open Shortest Path First” (OSPFv2, OSPFv3 is IPv6)

- OSPF is an “internal” gateway protocol
- Relies on IPv4 broadcasts to share routing information
- Ensures traffic is routed effectively (shortest-path)
- Based on Dijkstra’s algorithm (graph theory)

Problem 3 (15p)

Use Dijkstra’s shortest paths algorithm to find the shortest path from node A to all other nodes in the network graph in the figure below. Record the operation of the algorithm and the results filling up the Table below. Use the first column to record which node is moved from the red to black set in each step. Use the other columns to record the best distances from each node to node A at that step.



OSPFv2 Primer (cont.)

Every time a link changes states (e.g. comes up, is disconnected)...

- Routers send out “link-state advertisements” (LSAs) and each node independently re-runs Dijkstra’s algorithm
- All nodes will converge on the same view of the network
- A loop-free network is guaranteed (except during transience)
 - “Micro-loops” occur before the network has converged completely





Turning on OSPFv2 in FRR

We'll use OSPFv2 to organize our network. It's simple to turn on:

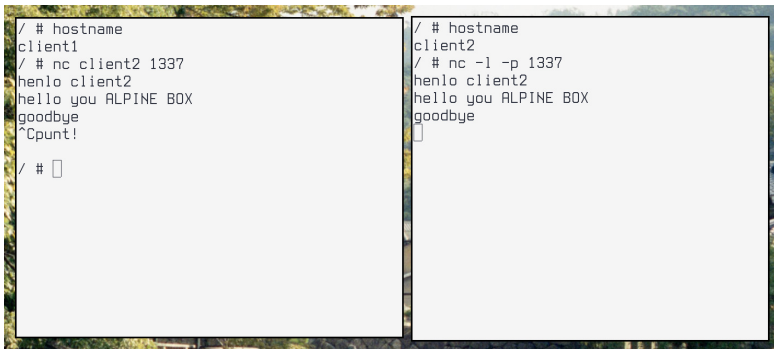
```
/ # hostname  
sw1  
/ # cat /etc/frr/ospfd.conf  
interface eth0  
    ip ospf area 0.0.0.0  
    ospf network point-to-point  
  
interface eth1  
    ip ospf area 0.0.0.0  
  
router ospf  
    passive-interface eth0  
/ # █
```

```
/ # hostname  
sw2  
/ # cat /etc/frr/ospfd.conf  
interface eth0  
    ip ospf area 0.0.0.0  
    ospf network point-to-point  
  
interface eth1  
    ip ospf area 0.0.0.0  
  
router ospf  
    passive-interface eth0  
/ # █
```

Figure: No static routes or hard-coded non-sense

Testing OSPFv2 Connectivity

After remoting in to both of the clients...



```
/ # hostname  
client1  
/ # nc client2 1337  
henlo client2  
hello you ALPINE BOX  
goodbye  
^Cpunt!  
  
/ # █
```

```
/ # hostname  
client2  
/ # nc -l -p 1337  
henlo client2  
hello you ALPINE BOX  
goodbye  
█
```

Figure: Connection success!

Testing OSPFv2 Connectivity (cont.)

The switches have figured out how to route traffic to the clients with very minimal effort:

```
sw1# show ip ospf route
===== OSPF network routing table =====
N    1.1.1.0/24          [10000] area: 0.0.0.0
      directly attached to eth1
N    10.1.1.0/24        [10000] area: 0.0.0.0
      directly attached to eth0
N    20.1.1.0/24        [20000] area: 0.0.0.0
      via 1.1.1.2, eth1

===== OSPF router routing table =====
===== OSPF external routing table =====

sw1#
```

Figure: Network routing table on sw1

Outline

- 1 Overview
 - What is Network Emulation?
 - Deploying Technology on an Emulated Network
- 2 A Survey of Existing Emulation Technologies
 - docker-topo
 - k8s-topo
 - Common Open Research Emulator
 - GNS-3
- 3 Using GNS-3
 - Generating a topology
 - Running an Actual Router
 - Turning on OSPFv2
- 4 Closing Remarks

Closing Remarks

We've seen how GNS-3 can be leveraged to network containers in order to explore and test WAN/LAN technologies

- We only covered super-simple OSPF today, 😞
- ...but this process could be generalized to test any routing protocol...
- We can use this to benchmark or develop our own protocols too!
 - Writing and benchmarking P2P software
 - Simulating torrent clusters
 - Adding protocol extensions to existing software

Closing Remarks (cont.)

ECE407 (Introduction to Computer Networking) covers topics like what we talked on today:

- OSI Model (pretty much every layer)
- Internal and External Gateway Protocols
- Wireshark and packet-switching networks

