FreeRouter in a Nutshell: A "Protocoland" routing platform for Open and Portable Carrier-Class Testbeds

Everson Scherrer Borges[1], Edgard da Cunha Pontes[1], Csaba Mate[2], Frederic Loui[3], Magnos Martinello[1], Moisés R. N. Ribeiro[1]

[1]Federal University of Espírito Santo (UFES)
Av. Fernando Ferrari, 514, Goiabeiras – 29.075-910 – Vitória – ES – Brazil

[2]RARE - Router for Academia Research
Education

[3]GÉANT - Pan-European Research and Education Networks

{everson.borges,edgard.pontes}@edu.ufes.br

Abstract. *An Open Source routing platform that uses a disaggregated model fully populated with carrier-class implementation of network protocols is discussed here. FreeRouter's architecture allows portability from emulated to testbed experiments (and on different hardware targets) is briefly explained. In order to demonstrate its ease of use, installation and entry-level examples are provided.*

Resumo. *Um roteador em software que propicia um plano de controle distribuído populado com um grande número de implementações profissionais de protocolos legados é aqui discutido. Brevemente discutimos também que a arquitetura particular do FreeRouter, a qual permite ainda a portabilidade de experimentos emulados para testes (e em diferentes alvos de hardware). Para demonstrar sua facilidade de uso, também são fornecidos exemplos de instalação e em nível básico de uso e configuração.*

1. Introduction

Network research is becoming more important since the Internet and other computer networks have a growing influence on the world. For this area of research, network testbeds are a crucial tool for experimentation. Network testbeds have been built to support the networking innovation. They can serve as platforms to experiment new services, as well as to enhance the use of services and protocols that are currently in production. Usually, the goal of testbeds is to enable a wide variety of experiments for the development, deployment, and validation of transformative solutions including clean-slate networking, protocol design and evaluation, content management, and in-network service deployment [Both et al. 2019].

The existing network testbeds usually offer a number devices in a laboratory room, or they may be spaded over wide-area the globe working as a distributed virtual laboratory with specific configurations. In contrast, nowadays you can have it all emulated, e.g. Mininet[1], in your personal computer. So, portability of experiments across emulated and

[1]http://www.mininet.org

physical world is a key feature for modern testbeds. The ideal scenario would be to debug functional/configuration/operational tests using your own personal computer with a trustworthy emulated version of the physical testbed, and only latter book physical resources of the testbed to do performance tests.

The experimenter's influence on this setup and its variables depends mainly on the testbed's architecture [Leung et al. 2014]. However, the potential innovation of the solutions is limited by the testbed architecture "ossification" often designed with its underlying closed technologies/boxes or even open approaches that require to re-implement all the legacy protocols lacking with production reality, losing portability of the experiments. Thus, decoupling control and data planes is a must in architectures for future-proof testbeds. Note that this also is related to portability since the same network solution (that could be used in core, edge and access levels) need to be realized using data planes suited both to the traffic volumes and to the cost constraints of each network domain.

Last but not least, a testbed based on open source is not only necessary for innovation and education purposes, but also for reducing cost and ensuring long-term sustainability. When a community is built around it, software reuse can improve learning curves and innovation processes. Nonetheless, ensuring diversity and trustworthiness of implementations for network protocols at a carrier-class level. Understood here as a well tested and proven implementation that is ready to inter-operate with other telecommunications industry equipment, carrier-class protocol implementation still is a challenge for the open source networking community. For instance, Quagga[2] has been adopted by the Linux Foundation in order to make the transition of a software router into commercial-grade level solution[3]. Note that full cycles involving development, deployment, and validation of open source often goes beyond the interest of academic groups, and startups are currently trying to figure out alternative long-lasting business models exploiting open platforms in products, from clouds to networks, such as Vixphy[4].

In this work, we briefly introduce **FreeRouter** (also known as freeRtr)[5] as a "protocoland" in the sense that its control plane, FreeRouter, stands out among similar software routers because it immense and outstanding protocol portfolio. In this "land of protocols" sandboxes can be effortlessly assembled and configured for user to play with them. Thus, we argue that FreeRouter should be considered not only when thinking of creating open and portable carrier-class testbed, but also for product prototyping in white boxes. FreeRouter is also defined as a "swiss army knife" due to its intrinsic capabilities of (re)encapsulating packets on multiple range of interfaces. However, for the networking community it not clear where FreeRouter stands in the current set of options for supporting open testbeds. Currently has been used and developed at RARE (Router for Academia, Research & Education RARE project)[6].

And its documentation set and entry level examples are scarce. Thus, herein we try to address those points, focusing on: i) historic context for control and data planes separation on open networking; and ii) disseminating FreeRouter to the research and ed-

[2] https://www.quagga.net/
[3] https://www.frrouting.org/
[4] http://www.vixphy.com.br
[5] http://freertr.org
[6] http://rare.freertr.org

ucation community via an installation and usage examples.

The remainder of this paper is as follows. **Section 2** discusses FreeRouter's architecture and particular features and relation with GÉANT GN4-3 (RARE). Step-by-step demonstrations for installing and using FreeRouter are provided in **Section 3**. Conclusions are drawn and future works are discussed in **Section 4**.

2. FreeRouter: Wisely Decoupling Control and Data Planes

There were early efforts from the Internet Engineering Task Force (IETF), such as the Forwarding and Control Element Separation (ForCES) in RFC 3746 in 2004 [Anderson; and Gopal 2004], before the paradigm of Software-Defined Networking (SDN) took the academic networking community by storm with OpenFlow in 2008 [McKeown et al. 2008]. Note that even changing the distributed to centralized approach regarding routing decisions was already considered before SDN via Path Computation Element (PCE)-Based Architecture in RFC 4655 in 2006 [Vasseur; and Ash 2006].

Nevertheless, both academy and industry communities did not succeed in bringing on a wave of innovation one could expect by such decoupling process of control and data planes. For the clients, on one hand, giving in the distributed approach to control planes would potentially decrease the intrinsic reliability already achieved by the internet. On the other hand, vendors were reluctant on opening access to "the goose that laid golden eggs" at data planes to their clients. In academy software routers were around for a while, but nothing would replace performance obtained by dedicated forwarding hardware-based forwarding engines. But, note that around 2015, a game changing initiative from a microprocessor vendor aim at openning source of a set of data plane libraries and network interface controller for direct access to operating system kernel to processes using Data Plane Development Kit (DPDK)[7].

From the perspective of academic groups, the clean-slate approach of SDN made a lot of sense and created room for research in this networking topic to bloom. But innovations, i.e., meeting real needs of carrier and enterprise mobile networks, lagged far behind the volumes of papers and prototype demonstrations. Note that even for conventional LAN/WAN networks, the roadmap for transitioning the legacy network to SDN was not clearly planned. For instance, clients did not feel comfortable with the idea of having now mocked instances of their legacy protocols being re-written from scratch and ported to "*logically centralized controller, which ideally can be physically distributed*" [Tootoonchian and Ganjali 2010][Spalla et al. 2016] and reached by a so-called "*secure channel*", a.k.a. TLS-based asymmetrical encryption over unencrypted TCP connections, established between a controller and an OpenFlow switch. Moreover, only limited implementations of OpenFlow specifications were supported by most vendors[8] and, in the end, performance (e.g., throughput, latency and scalability) were also very heterogeneous across vendors and OpenFlow versions, e.g., [Costa et al. 2021], reflecting the lack of common ground for programmability at ASIC level.

In order to devise an open source business model for SDN related technologies, the Open Networking Foundation[9] (ONF) was created in 2011 and focused on carrier, cloud

[7] `https://www.dpdk.org`

[8] `https://blog.ipspace.net/2016/12/q-vendor-openflow-limitations.html`

[9] `http://www.opennetworking.org`

and mobile operators needs. A huge leap into network function virtualization (NFV) and its use in real networks was made by the SDN-enabled software switch projects around OpenFlow. The Open virtual Switch (OvS) was incorporated by The Linux kernel into the kernel in 2012; and it is now a key element in cloud computing software platforms and virtualization management systems such as OpenStack[10]. The potential of solve complex orchestration processes between cloud and physical resources can be found here [Dominicini et al. 2017], while dynamic reconfigurations required by Service Function Chaining (SFC) could be provided by nesting multiple instances of OvS [Dominicini et al. 2020].

Despite a relative success in niches like campus and data center networks, and also for WAN adoption by Google [Jain et al. 2013], SDN had its days, and after a decade on the road OpenFlow could not deliver on its promises. Nevertheless, network programmability is not dead yet. Passing on a legacy baton, in 2014 the Programming Protocol-Independent Packet Processors (P4) dives into dataplanes' ASICs and provides expressivity levels well beyond match-action label-based programmability enabled by OpenFlow. Which begs the question: is P4 yet another window of opportunity for innovation that would require rewriting the whole set of legacy protocols of the internet?

In that context, FreeRouter brings an unique contribution in tackling some of the weakness discussed above regarding strategies for splitting control and forwarding functions. Open source since 2012, more recently FreeRouter framework was combined with state-of-art data planes, such as DPDK and P4, to produce a decoupled architecture that sits at the sweet spot by keeping a distributed architecture for control plane with tailored south-bound interfaces.

Figure 1 provides a simplified diagram for FreeRouter's framework and its use on RARE where the data plane portability issue is addressed. Note that testbed users are as agnostic as possible of the dataplanes in use, and interact with testbed elements via standard industry-like CLI interfaces for configuration. Carrier-grade open source implementations a large set of protocols are available. Hidden from users, south-bound interfaces are fully responsible for translating control plane of legacy protocols (e.g., forward information base and into pro communicate them to diverse types data planes.

As far as implementation is concerned, FreeRouter is a user-level application written in Java. It consists of a core aplication rtr.jar is working as a control plane software that natively relies on UNIX UDP socket. One unprivileged JVM process (per Virtual Device Context: VDC) that does everything a router does and communicates with the world around it over UDP sockets. Ethernet packets placed back and forth to UDP socket with socat linux utility. Regarding VDC, other JVM routers or QEMU/KVM images can be started, UDP socket passes traffic between them. There is also a purpose built libpcap based C code which signals ethernet up/downs and outperforms socat about 80 %. A purpose built C code for async HDLC framing. Other helpers or table dumps (to ASICs, OpenFlow, whatever needed) are easily achievable.

2.1. Carrier-Class Protocol Implementation: Testing and Releasing Procedures

In order to support legacy applications, FreeRouter brings along a densily populated control plane able to handle: **i)** diverse encapsulation; **ii)** cripto; **iii)** both table-based and also

[10] `http://www.openstack.org`

lable-based at L2 and L3 forwarding; **iii)** diverse tunneling; **iv)** both IGP and EGP routing protocols; and **v)** Policy-Based Routing services, NAT, QoS, and other services. More detailed list of protocols on different layers (and their combinations) implemented and tested at the moment (on different data planes) are available in RARE project[11].

Before releasing a new protocol (or a new version) feature oriented tests for every given feature of a given supported protocol. Careful log outputs from the routers are taken for for further analysis. An object oriented language designs combination/permutation patterns bigger test cases to test some features together. Moreover, big buck of tests are dedicated to *interop tests*, where we test the features against some prominent vendor virtual images. Finally we have the data plane test cases, where we test the given data plane is capable of performing the given forwarding feature.

2.2. Portability to Different Dataplanes and GÉANT GN4-3 RARE Project

FreeRouter allows users to spawn an unlimited amount of router processes on the same host, and interconnect them via UNIX UDP sockets in order to implement a topology and emulate an entire network just like Mininet. Moreover, as the first portability step, white-boxe hosts running FreeRouter can turn the emulated network into operational elements.

Regarding performance concerns, Linux's raw packet handler needs about same number of cpu cycles as FreeRouter's forwarding code. JVM7+ uses AES extension of CPUs, if available, so crypto can perform very well. JVM optimizes branches in real time so unused code gets optimized out on the fly. For example, NIIF/Hungarnet whitebox implementation described in FreeRouter's webpage was able to outperform a vendor's BGP stack.

It is important to highlight that FreeRouter is taken even further into the RARE project (Router for Academia, Research Education)[12]. It is a GÉANT 4th programme on routing software platform for Research and Education use cases. Although initially focused on P4 behavioral language is used to describe the packet processing behavior of RARE data plane and communication interface between the control plane and data plane: Interface compliant to P4Runtime specification ensure this function. Advanced data plane portability is beyond our scope here, more information can be found at RARE's webpage.

The FIG. 1 highlights a remarkable capability that differentiates FreeRouter from other existing applications, these work only as control plane routing, while FreeRouter makes it possible to communicate with data plane being able to communicate with different data planes like DPDK, XDP, P4 among others[Loui et al. 2022]. This feature makes it possible to prototype different protocols, as well as perform tests and experiments.

3. Demonstrating FreeRouter: a step-by-step tutorial

This section describes the FreeRouter routing platform. The only requirement to run Freerouter is the latest version of Java Runtime Environment (JRE)[13].

[11] `https://wiki.geant.org/display/RARE/Complete+feature+list`

[12] `https://wiki.geant.org/display/RARE/Home`

[13] `https://www.liquidweb.com/kb/how-to-install-java-on-ubuntu-windows-and-macos/`

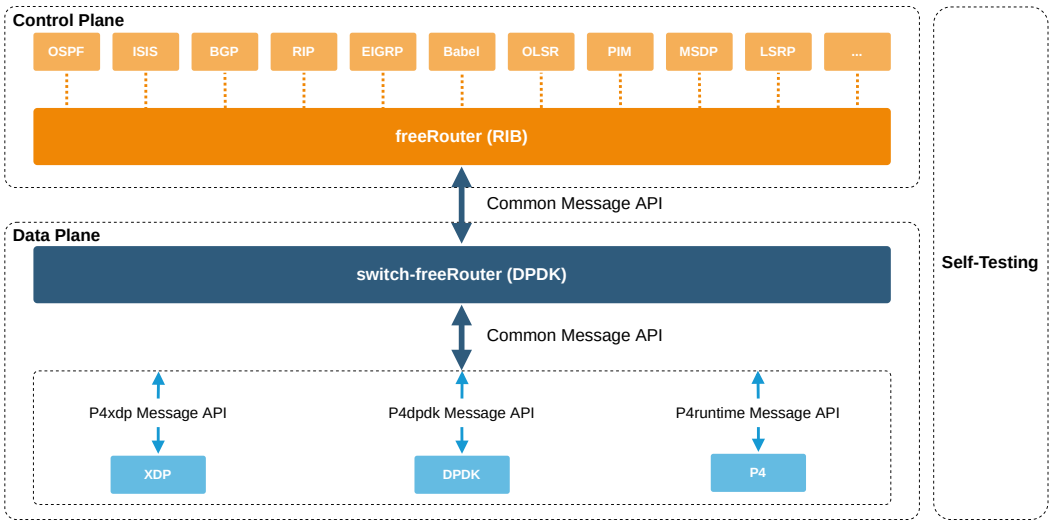


Figure 1. FreeRouter High Level Architecture

3.1. Configuration of routers and their interconnection network

FreeRouter needs two files in order to run properly:

1. A hardware definition file;
2. A software configuration file.

3.1.1. Hardware Configuration

This file covers the hardware router definition:

- interfaces definition;
- external port translation to FreeRouter port namespace;
- external process launched and watched by FreeRouter.

An example of hardware configuration file can be seen at TABLE 1 in **Section 3.3.1**.

3.1.2. Software Configuration

The software definition file brings all the configurations of interfaces, services, VRFs, routable protocols, and routing protocols, among other standard configurations of a switch/router, with commands similar to the leading suppliers.

An example of software configuration file can be seen at TABLE 2 in **Section 3.3.1**.

3.2. Launching FreeRouter

After installing the necessary prerequisites and understanding how the hardware and software files work, it's time to run the FreeRouter.

```
java -jar <path>/rtr.jar routersc <path>/router-hw.txt <path>/
   router-sw.txt
```

3.3. Experimentation

By supporting a large number of standardized routing protocols (e.g. OSPF, RIP, BGP) as well as Label Switching Protocols such as MPLS, FreeRouter forms a "protocoland" for experimenters. Note that this is in clear contrast with Mininet emulation tool, where SDN versions of such protocols would be needed in order to mock such protocols, in case interaction with legacy protocols are sought. Thus, it is important to highlight that FreeRouter's distributed architecture and carrier-class protocol implementation is an upper hand when compared to Mininet's control and data plane SDN-based separation. As a result, FreeRouter provides a step towards a **near production deployment** and integration with other legacy protocols with programmable data planes like no other emulation tool.

3.3.1. Dynamic Full-Mesh Routing with OSPF

In order to introduce Freerouter functionalities, we assume as an use case, an autonomous system that needs to setup a routing policy in its intra-domain. The OSPF[14] is the protocol selected to run in a full-mesh topology. The reason for its choice is the practical applicability, easily and well-known dynamic routing protocol for both IPv4 and IPv6 networks to keep the service connectivity. In this configuration, a template feature of FreeRouter was used to define the router network interfaces in a simpler and faster way.

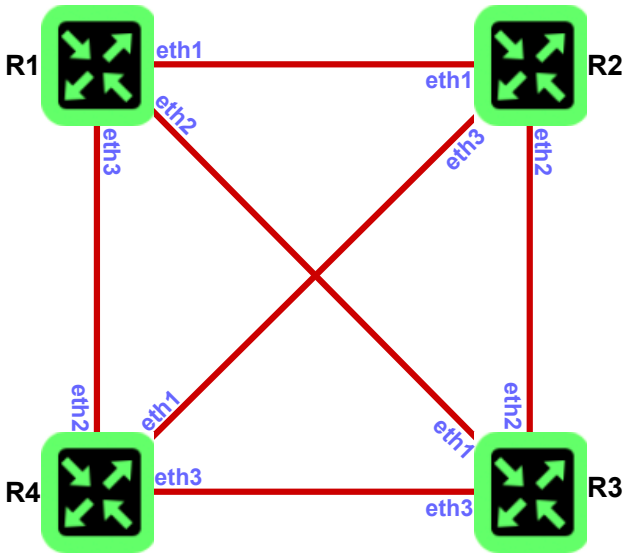


Figure 2. Full-Mesh Topology

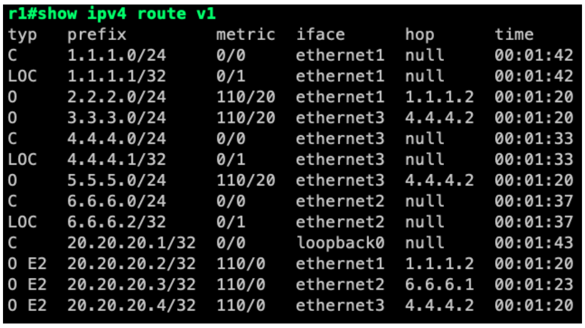
To perform the entire experiment presented in FIG. 2, it is necessary to have a hardware file and a software file for each router, covering all the interconnection between the nodes (hardware file), as well as the definitions of each router (software file). In tables 1 and 2, the complete configurations of router 1 are described. To know all the details of hardware and software files of complet set of routers (R_2, R_3, R_4), please access GitHub's repository[15].

[14] https://datatracker.ietf.org/doc/html/rfc2328
[15] https://github.com/eversonscherrer/wtestbeds2022

r1-hw.txt
int eth1 eth 0000.1111.0001 127.0.0.1 26011 127.0.0.1 26021
int eth2 eth 0000.1111.0002 127.0.0.1 26012 127.0.0.1 26031
int eth3 eth 0000.1111.0003 127.0.0.1 26013 127.0.0.1 26042
tcp2vrf 1123 v1 23

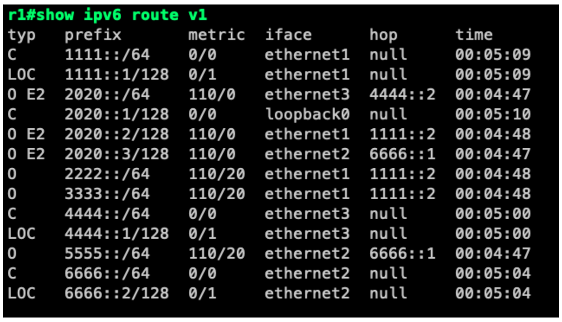
3.3.2. Troubleshooting

As it can be seen in FIG. 3a and FIG. 3b, the routing table for each router is shown to IPv4 and IPv6. For instance, the command `route v1` shows the routing table in a similar way with commands used in Cisco, Juniper, etc.



```
r1#show ipv4 route v1
typ   prefix         metric  iface      hop      time
C     1.1.1.0/24     0/0     ethernet1  null     00:01:42
LOC   1.1.1.1/32     0/1     ethernet1  null     00:01:42
O     2.2.2.0/24     110/20  ethernet1  1.1.1.2  00:01:20
O     3.3.3.0/24     110/20  ethernet3  4.4.4.2  00:01:20
C     4.4.4.0/24     0/0     ethernet3  null     00:01:33
LOC   4.4.4.1/32     0/1     ethernet3  null     00:01:33
O     5.5.5.0/24     110/20  ethernet3  4.4.4.2  00:01:20
C     6.6.6.0/24     0/0     ethernet2  null     00:01:37
LOC   6.6.6.2/32     0/1     ethernet2  null     00:01:37
C     20.20.20.1/32  0/0     loopback0  null     00:01:43
O E2  20.20.20.2/32  110/0   ethernet1  1.1.1.2  00:01:20
O E2  20.20.20.3/32  110/0   ethernet2  6.6.6.1  00:01:23
O E2  20.20.20.4/32  110/0   ethernet3  4.4.4.2  00:01:20
```

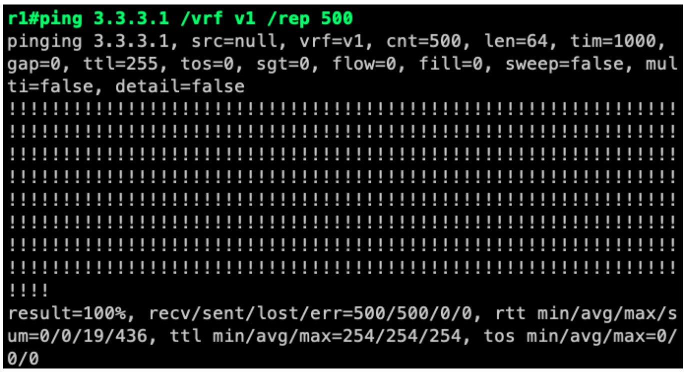
(a) Routes to router r1 (IPv4)



```
r1#show ipv6 route v1
typ   prefix       metric  iface      hop      time
C     1111::/64    0/0     ethernet1  null     00:05:09
LOC   1111::1/128  0/1     ethernet1  null     00:05:09
O E2  2020::/64    110/0   ethernet3  4444::2  00:04:47
C     2020::1/128  0/0     loopback0  null     00:05:10
O E2  2020::2/128  110/0   ethernet1  1111::2  00:04:48
O E2  2020::3/128  110/0   ethernet2  6666::1  00:04:47
O     2222::/64    110/20  ethernet1  1111::2  00:04:48
O     3333::/64    110/20  ethernet1  1111::2  00:04:48
C     4444::/64    0/0     ethernet3  null     00:05:00
LOC   4444::1/128  0/1     ethernet3  null     00:05:00
O     5555::/64    110/20  ethernet2  6666::1  00:04:47
C     6666::/64    0/0     ethernet2  null     00:05:04
LOC   6666::2/128  0/1     ethernet2  null     00:05:04
```

(b) Routes to router r1 (IPv6)

When checking the connectivity from router `R1` to router `R3` with the following command `ping 3.3.3.1 /vrf v1 /repeat 500`, it's possible to observe through FIG. 5a that all interfaces are connected and router `R1` is forwarding traffic over the `eth3`.



```
r1#ping 3.3.3.1 /vrf v1 /rep 500
pinging 3.3.3.1, src=null, vrf=v1, cnt=500, len=64, tim=1000,
gap=0, ttl=255, tos=0, sgt=0, flow=0, fill=0, sweep=false, mul
ti=false, detail=false
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!
result=100%, recv/sent/lost/err=500/500/0/0, rtt min/avg/max/s
um=0/0/19/436, ttl min/avg/max=254/254/254, tos min/avg/max=0/
0/0
```

Figure 4. Testing connectivity

By provoking a failure recovery test, the `eth3` interface is `shutdown`, so it can be seen through FIG. 5b, that the traffic change to forward to the `eth1` interface. Another failure was provoked now `shutdown` the interface `eth1`, again the FreeRouter performs the fault recovery forwarding the traffic to the `eth2`, highlighted by FIG. 5c.

As can be seen from FIG. 6 only the `eth2` interface is `up`, even so, the FreeRouter using OSPF routing protocol allows the recovery of routes, keeping the topology working.

Besides learning and prototyping, FreeRouter enables easy deployment in testbeds. For instance, in order to create you own testbed (and deploy the experiment

r1-sw.txt

```
hostname r1
!
vrf definition v1
 exit
!
router ospf4 1
 vrf v1
 router-id 10.1.1.1
 area 0 enable
 redistribute connected
 exit
!
router ospf6 1
 vrf v1
 router-id 10.6.1.1
 area 0 enable
 redistribute connected
 exit
!
interface template1
 no description
 lldp enable
 vrf forwarding v1
 ipv4 address dynamic dynamic
 router ospf4 1 enable
 shutdown
 no log-link-change
 exit
!
int lo0
 vrf for v1
 ipv4 addr 20.20.20.1 /32
 ipv6 addr 2020::1 /128
 exit
!
interface ethernet1
 description r1@eth1 -> r2@eth1
 vrf forwarding v1
 ipv4 address 1.1.1.1 /24
 ipv6 address 1111::1 /64
 template template1
 router ospf6 1 enable
 no shutdown
 no log-link-change
 exit
!
interface ethernet2
 description r1@eth2 -> r3@eth2
 vrf forwarding v1
 ipv4 address 6.6.6.2 /24
 ipv6 address 6666::2 /64
 template template1
 router ospf6 1 enable
 no shutdown
 no log-link-change
 exit
!
interface ethernet3
 description r1@eth3 -> r3@eth1
 vrf forwarding v1
 ipv4 address 4.4.4.1 /24
 ipv6 address 4444::1 /64
 template template1
 router ospf6 1 enable
 no shutdown
 no log-link-change
 exit
!
server telnet tel
 security protocol telnet
 exec timeout 10000000
 exec colorize prompt
 exec logging
 no exec authorization
 no login authentication
 login logging
 vrf v1
 exit
!
end
```

```
r1#show interfaces traffic
interface  state  tx      rx      drop
template1  admin  0       0       0
loopback0  up     0       0       0
ethernet1  up     0       0       0
ethernet2  up     0       0       0
ethernet3  up     283272  283206  0
```

(a) Traffic all interfaces up

```
r1#show interfaces traffic
interface  state  tx      rx      drop
template1  admin  0       0       0
loopback0  up     0       0       0
ethernet1  up     199208  199122  0
ethernet2  up     0       0       0
ethernet3  admin  0       0       0
```

(b) Traffic interfaces, eth3 down

```
r1#show interfaces traffic
interface  state  tx      rx      drop
template1  admin  0       0       0
loopback0  up     0       0       0
ethernet1  admin  0       0       0
ethernet2  up     195426  195426  0
ethernet3  admin  0       0       0
```

(c) Traffic interfaces, only eth2 up

```
r1#show ipv4 route v1
typ    prefix          metric  iface      hop      time
O      1.1.1.0/24      110/30  ethernet2  6.6.6.1  00:00:49
O      2.2.2.0/24      110/20  ethernet2  6.6.6.1  00:00:49
O      3.3.3.0/24      110/30  ethernet2  6.6.6.1  00:00:49
O      4.4.4.0/24      110/30  ethernet2  6.6.6.1  00:00:49
O      5.5.5.0/24      110/20  ethernet2  6.6.6.1  00:01:32
C      6.6.6.0/24      0/0     ethernet2  null     00:05:23
LOC    6.6.6.2/32      0/1     ethernet2  null     00:05:23
C      20.20.20.1/32   0/0     loopback0  null     00:05:24
O E2   20.20.20.2/32   110/0   ethernet2  6.6.6.1  00:00:49
O E2   20.20.20.3/32   110/0   ethernet2  6.6.6.1  00:05:11
O E2   20.20.20.4/32   110/0   ethernet2  6.6.6.1  00:00:49
```

Figure 6. Failure recovery by OSPF

discussed above), one may use containers, Virtual Machines, or even physical servers for hosting the four different instances of FreeRouter for representing R1-R4. The only modification from the emulated version is to bind the interfaces of such instances with the corresponding Ethernet interfaces, (either physical or virtual) to unleash FreeRouter's the full potential[16]. Moreover, R1-R4 can be also Intel Tofino P4 hardware. In that case, installation guide and how to perform the data plane software installation is also available.[17]

4. Conclusion

This work provided a short introduction to FreeRouter and provided arguments in favor of its use when creating open and portable carrier-class testbed. A historic study on open networking was presented and FreeRouter's control and data plane unique separation strategy context put into context. In order to disseminate FreeRouter to the research and education community, installation and entry-level configurations were presented.

However, FreeRouter has much to improve and welcomes the networking community worldwide to help developing: **i)** networking programming/configuration by examples; **ii)** strategies to make FreeRouter easy to use also for fostering innovation (i.e, how can one develop new protocols without close assistance's of FreeRouter's only developer?); **iii)** clarification for the architecture elements/components; **iv)** formalization of protocol test for homologation processes; **v)** extensive studies for outlining competitive advantages/disadvantages in comparison to other solutions; **vi)** exhaustive data plane performance evaluation; and **vii)** studies on proving strong consistency for table processing at data planes as a means of representing legacy control plane protocols.

5. Acknowledgments

This study received funds from CNPq, FAPESP, FAPES, CTIC, and RNP.

[16] https://docs.freertr.org/guides/getting-started/003-unleash/
[17] https://https://docs.freertr.org/guides/installation/

References

Anderson;, L. Y. R. D. T. and Gopal, R. (2004). Rfc 3746: Forwarding and control element separation (forces) framework. https://datatracker.ietf.org/doc/html/rfc3746.

Both, C., Guimaraes, R., Slyne, F., Wickboldt, J., Martinello, M., Dominicini, C., Martins, R., Zhang, Y., Cardoso, D., Villaca, R., Ceravolo, I., Nejabati, R., Marquez-Barja, J., Ruffini, M., and DaSilva, L. (2019). Futebol control framework: Enabling experimentation in convergent optical, wireless, and cloud infrastructures. *IEEE Communications Magazine*, 57(10):56–62.

Costa, L. C., Vieira, A. B., de Britto e Silva, E., Macedo, D. F., Vieira, L. F., Vieira, M. A., da Rocha Miranda, M., Batista, G. F., Polizer, A. H., Gonçalves, A. V. G. S., Gomes, G., and Correia, L. H. (2021). Openflow data planes performance evaluation. *Performance Evaluation*, 147:102194.

Dominicini, C. K., Vassoler, G. L., Meneses, L. F., Villaca, R. S., Ribeiro, M. R. N., and Martinello, M. (2017). Virtphy: Fully programmable nfv orchestration architecture for edge data centers. *IEEE Transactions on Network and Service Management*, 14(4):817–830.

Dominicini, C. K., Vassoler, G. L., Valentim, R., Villaca, R. S., Ribeiro, M. R., Martinello, M., and Zambon, E. (2020). Keysfc: Traffic steering using strict source routing for dynamic and efficient network orchestration. *Computer Networks*, 167:106975.

Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., and Vahdat, A. (2013). B4: Experience with a globally-deployed software defined wan. New York, NY, USA. Association for Computing Machinery.

Leung, V. C., Chen, M., Wan, J., and Zhang, Y. (2014). *Testbeds and Research Infrastructure: Development of Networks and Communities: 9th International ICST Conference, TridentCom 2014, Guangzhou, China, May 5-7, 2014, Revised Selected Papers*, volume 137. Springer.

Loui, F., Mate, C., Gall, A., and Wisslé, M. (2022). Project overview: Router for academia research & education (rare).

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. 38(2).

Spalla, E. S., Mafioletti, D. R., Liberato, A. B., Ewald, G., Rothenberg, C. E., Camargos, L., Villaca, R. S., and Martinello, M. (2016). Ar2c2: Actively replicated controllers for sdn resilient control plane. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 189–196.

Tootoonchian, A. and Ganjali, Y. (2010). Hyperflow: A distributed control plane for openflow. USA. USENIX Association.

Vasseur;, A. F. J.-P. and Ash, J. (2006). Rfc 4655: A path computation element (pce)-based architecture. https://datatracker.ietf.org/doc/html/rfc3746.