FABRIC Testbed from the Eyes of a Network Researcher

Edgard da Cunha Pontes[1], Magnos Martinello[15], Cristina K. Dominicini[2], Marcos Schwarz[4], Moises Ribeiro[1], Everson S. Borges[12], Italo Brito[3], Jeronimo Bezerra[3], Marinho Barcellos[5]

[1]Universidade Federal do Espírito Santo (UFES)
Av. Fernando Ferrari, 514, Goiabeiras – 29.075-910 – Vitória – ES – Brasil

[2]Federal Institute of Espirito Santo - IFES - Brazil

[3]Center for Internet Augmented Research and Assessment –
Florida International University (FIU) Miami, Florida, US

[4]Rede Nacional de Pesquisa (RNP) – Campinas, SP – Brasil

[5]University of Waikato – Hamilton, – New Zealand

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

marcos.schwarz@rnp.br

Abstract. *The purpose of this paper is twofold. The first is to show how to deploy routing protocols in the FABRIC national-scale testbed, presenting examples step-by-step from the perspective of a network researcher who needs to book a slice composed of a topology with multiple paths, diversity of links, nodes with specific network interfaces (shared or/and programmable NICs) that form the basis to build a real-world network experiment. The second is to use the available tools ("metaresearch" API) to demonstrate how reproducible is the experimentation process, taking the evaluation of OSPF failure recovery and traffic throughput over dedicated smartNICS as use cases.*

Resumo. *O objetivo deste artigo é duplo. O primeiro é mostrar como implantar protocolos de roteamento no testbed FABRIC em escala nacional, apresentando exemplos passo a passo sob a perspectiva de um pesquisador de redes que precisa reservar uma fatia composta por uma topologia com múltiplos caminhos, diversidade de links, nós com interfaces de rede específicas (NICs compartilhadas e/ou programáveis) que formam a base para construir um experimento de rede do mundo real. O segundo é usar as ferramentas disponíveis ("metaresearch" API) para demonstrar o quão reprodutível é o processo de experimentação, tendo como casos de uso a avaliação da recuperação de falhas com o OSPF e a taxa de transferência do tráfego com smartNICS dedicadas.*

1. Introduction

Testbeds are experimental platforms designed to facilitate the development, testing, and evaluation of new technologies and applications in various domains. They typically consist of a collection of hardware and software components, including network infrastructure, computing resources, and data storage. Usually, the goal of testbeds is to enable a

wide variety of experiments for the development, deployment, and validation of innovative solutions including clean-slate networking, protocol design and evaluation, cybersecurity, and in-network service deployment [Both et al. 2019].

Experiments involving networks are difficult to run and depend on the testbed's architecture and tools [Borges et al. 2022b]. Moreover, performing real-world experiments is considered a prerequisite for a true validation of a protocol, and it requires both *repeatability* and *reproducibility*. Repeatability requires that the same experiment runs in similar conditions several times by the experimenter, producing equivalent results. This is important to guarantee that experimental results were fine. Reproducibility is guaranteed if an experiment can be replicated under differing conditions, while providing sufficiently similar results. This is essential to prove that the scientific proposal is robust across various environments and not only in a particular setup. It is also crucial for allowing researchers to reproduce experiments, build upon them, and compare their results with previous work.

FABRIC testbed [Baldin et al. 2019] has been designed to provide a unique national research infrastructure to enable cutting-edge and exploratory research at scale in networking. They offer "data artifacts" collected by experimenters in distributed storage facilities that can be made available to the wider community. This "metaresearch" is a feature to support the experiment's reproducibility, by giving researchers a trove of data on user actions captured from the experimentation for learning/analytics applications. This feature needs to be tested/consolidated from the perspective of a network researcher. Typically, an experimentation plan requires a booked slice. The slice is an instance of resources (computing nodes, routers, links, and a variety of configurations) to be used in experiments, collecting and extracting the results.

In this paper, we take the position of a network researcher who has a specific experimentation requirement. She/he wants a topology composed by multiple paths, with a diversity of links in terms of capacity and propagation delays, and nodes with specific network interfaces (shared or/and programmable NICs) that form the elements of his/her own network experiment. Keeping that requirement in mind, our goal is **i)** to show how to deploy routing protocols in the FABRIC national-scale testbed, presenting examples step-by-step from the perspective of a network researcher, but also **ii)** to use the available tools ("metaresearch" API) to demonstrate how reproducible is the experimentation process, taking the OSPF failure recovery and traffic throughput over dedicated smartNICs as use cases. All the examples are open and available to the research and education community, and are shared in a GitHub[1] with jupitter notebook.

The remainder of this paper is as follows. **Section 2** discusses FABRIC view and its particular features for slice definition, booking a set of components, configuration deployment and setup automation. We run experiments using the testbed library and its support in **Section 3**. Conclusions are drawn and future works are discussed in **Section 4**.

2. FABRIC view from the eyes of a network researcher

Looking at the state-of-the-art of experimental testbeds, there has been a large number of projects that provide experimental facilities for networking experimentation. The GENI project was pioneer to build an open and distributed infrastructure at-scale for research

[1] `https://github.com/edgardcunha/wtestbeds2023`

and education [Berman et al. 2014]. The OFELIA project [M. et al. 2014] created an experimental facility that was based on SDN/OpenFlow to control the network environment. With a similar goal, Japan had the RISE project, which was an OpenFlow-based research infrastructure for large-scale network experiments on the Japan Gigabit Network [Huang et al. 2017]. In Brazil, the FIBRE project [Salmito et al. 2014] built federated testbeds to provide a large-scale research platform. Also, FUTEBOL project [Both et al. 2019] was conceived to enable experimental research on optical, wireless, and cloud convergence that has been an enabler to the recent 5G networks architecture [2].

The FABRIC's ambition is to become a widely distributed instrument both for computer science research and other science domains that want to explore faster and more capable distributed computational and data infrastructures. It has been designed to be highly programmable, with tera-bits-per-second core network nodes interconnecting existing facilities e.g. GENI [Berman et al. 2014], and future resources such as Open Cloud [Zink et al. 2021], campus networks and clusters, national HPC and data facilities, dedicated to experimentation but running in parallel to production networks.

From the eyes of a researcher in networking, FABRIC's core nodes will be deployed with a special footprint of the Department of Energy's Energy Sciences Network (ESnet), complemented by programmable edge nodes on campuses and national facilities to enable network programmability. **Figure 2** shows the FABRIC testbed topology with a diversity of links in terms of capacity, propagation delays and multiplicity of paths. Therefore, it is a natural step forward to extend our previous work [Borges et al. 2022b], deploying routing protocols in order to experiment different levels of network programmability at a national-scale testbed by using the "metaresearch" provided in FABRIC framework.

In order to run a reproducible experiment, we need first of all to get a slice from the testbed, deploy all the required configuration, be able to automate the experiment components to make it easily repeatable and then run it many times, so that there is a sequence of steps to be followed: slice definition, nodes and network configuration allowing automation of the experiment components, test of network connectivity to all the allocated nodes, enabling nodes to become routers for networking purposes, make use of network programmability support at the devices to accelerate packets forwarding (e.g. activate DPDK) and to offload P4 codes at smartNiCS or P4 switches, and finally to execute all the experiments to get performance results.

2.1. Slice definition: requirements and allocation

After the project and users are properly registered, it is necessary to create asymmetric access keys. The testbed security policy requires registration and management of these keys. There are three types of keys to access the testbed: bastion key, sliver key and project token.

[2] `https://porvir-5g-project.github.io`

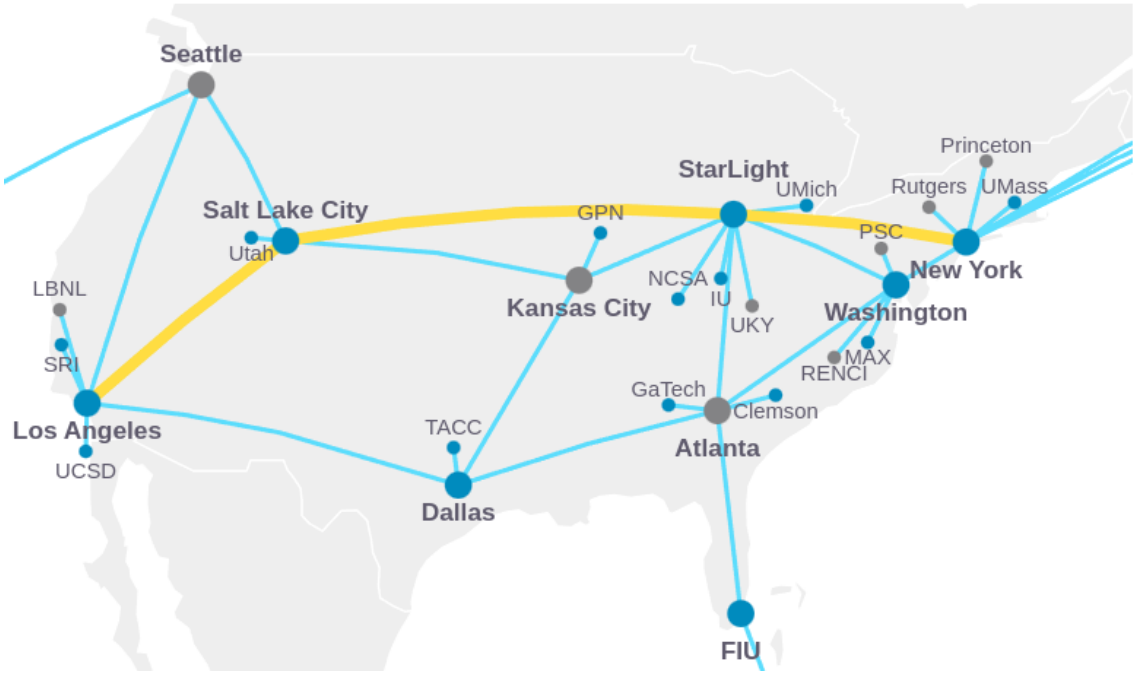

Figure 1. Topology of FABRIC Testbed

2.1.1. Booking a slice

To book a slice in FABRIC Testbed and use its metaresearch functionalities, the experimenter needs to import the "FABlib library" (v. 1.4.3) that provides its API primitives. For example, one may query for available testbed resources and settings by using `fablib.list_sites()` and then get 3 random sites that are available for experimentation with the command `fablib.get_random_sites(count=3)`. There is also a geographic location parameter to select sites located between Los Angeles and New York as shown in Figure 2.

```
from fabrictestbed_extensions.fablib.fablib import FablibManager as fablib_manager

fablib = fablib_manager()
los_angelis_lat_long=(34.049043, -118.259476)
new_york_lat_long=(43.453157655429585, -76.53364473809769)

try:
    sites = fablib.get_random_sites(count=3, filter_function=lambda x:
                                        x['nic_connectx_5_available'] >= 2 and
                                        x['cores_available'] >= 2 and
                                        x['disk_available'] >= 10 and
                                        x['location'][1] > los_angelis_lat_long[1] and
                                        x['location'][1] < new_york_lat_long[1]
                                    )
    print(f"Available Sites: {sites}")
except Exception as e:
    print(f"Exception: {e}")
```

```
Available Sites: ['MICH', 'UTAH', 'STAR']
```

Figure 2. Get 3 random sites with geographic location filter

New components can be added to the slice. For example, a dedicated smartNIC (command `add_component()` or a L2 link, see **Figure 3**. After all the requirements

specified for the experiment components, the request of the slice can be submitted. Note that the OS can be customized for each slice node. It is even possible to use Docker containers on the nodes. Currently available operating systems are: CentOS Stream (8 e 9), CentOS (7 e 8), Debian (10 e 11), Fedora (35, 36 e 37), Rocky Linux (8), Ubuntu (18.04, 20.04 , 21.04 and 22.04), as well as custom versions of Rocky Linux and Ubuntu. In this example, we use Debian 10 (`default_debian_10`).

As a result, the slice's attributes and visualization are shown in **Figure 4**. For the sake of simplicity, a tree-node ring will be here used to demonstrate network dynamic routing reacting to node failure, which was explored in the experiments.

```
slice_name = 'OSPF_Routing_Topology'
try:
    slice = fablib.new_slice(name= slice_name)

    nodes, nics = [], []

    for i, name in enumerate(sites):
        nodes.insert(i, slice.add_node(name=f'r{i+1}', site=name, image='default_debian_10'))

    for node in nodes:
        nics.insert(i, node.add_component(model='NIC_Basic', name='nic1').get_interfaces()[0])
        nics.insert(i+1, node.add_component(model='NIC_Basic', name='nic2').get_interfaces()[0])

    net1 = slice.add_l2network(name='net1', interfaces=[nics[0], nics[3]])
    net2 = slice.add_l2network(name='net2', interfaces=[nics[2], nics[5]])
    net3 = slice.add_l2network(name='net3', interfaces=[nics[4], nics[1]])

    slice.submit()
except Exception as e:
    print(f"Exception: {e}")
```

Figure 3. Node and component configuration

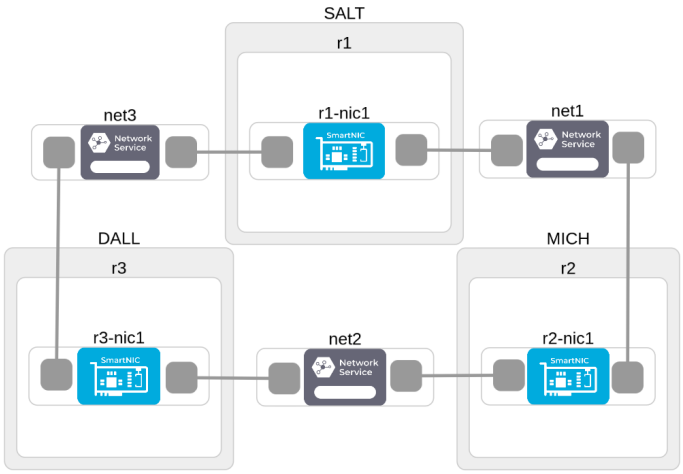

Figure 4. Slice visualization: ring topology connected with dedicated smartnics

2.1.2. Components configuration and automation

Once the slice has been allocated, we need to install specific software and set all the parameters. For example, the operation system must be chosen and the network interfaces must be properly configured.

Debian OS Upgrade was mandatory to use dedicated NICs (SmartNICs) on FABRIC[3]. Automation tools such as Ansible are very useful to reduce the time to install and deploy all the configurations (more details can be seen in the example of **Figure 5**). It is worth noting that a configuration error is shown by purpose when accessing router `R2`. This automation process via Ansible is helpful in setting the experimental nodes bring some warnings if the OS packages were correctly updated reporting eventual errors.

```
# Upgrade from debian 10 'buster' to 11 'bullseye'
!ansible-playbook -i inventory.yml full_upgrade.yml

PLAY [all] *********************************************************************

TASK [Gathering Facts] *********************************************************
ok: [r1]
ok: [r3]
fatal: [r2]: UNREACHABLE! => {"changed": false, "msg": "Failed to connect to the host via ssh: Connection timed out
during banner exchange", "unreachable": true}

TASK [Updating repositories 'apt-get update'] **********************************
ok: [r1]
ok: [r3]

TASK [Upgrading all packages 'apt-get upgrade'] ********************************
skipping: [r1]
skipping: [r3]

TASK [Editing Debian repository from 10 'buster' to 11 'bullseye'] *************
skipping: [r1] => (item={'From': 'buster', 'To': 'bullseye'})
skipping: [r1]
skipping: [r3] => (item={'From': 'buster', 'To': 'bullseye'})
skipping: [r3]

TASK [Editing security repository format (bullseye/updates > bullseye-security)] ***
skipping: [r1]
skipping: [r3]

TASK [Updating repositories 'apt-get update'] **********************************
skipping: [r1]
skipping: [r3]

TASK [Full-upgrading all packages 'apt-get full-upgrade'] **********************
skipping: [r1]
skipping: [r3]

TASK [Checking if a system reboot is required] *********************************
skipping: [r1]
skipping: [r3]

TASK [Rebooting the system due to kernel update] *******************************
changed: [r1]
changed: [r3]

PLAY RECAP *********************************************************************
r1                         : ok=3    changed=1    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0
r2                         : ok=0    changed=0    unreachable=1    failed=0    skipped=0    rescued=0    ignored=0
r3                         : ok=3    changed=1    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0
```

Figure 5. Ansible tasks to upgrade Debian

The IP addressing is defined and assigned by creating a set of subnets interconnected by their links that compose the network topology (see **Figure 6**). We pick the subnet for the routing links. Each routing link connects a pair of routers. Although these links have exactly two interfaces, we choose a /24 subnet for easy readability. For each link we create a subnet and a list of available IPs for that subnet. These will be used later to configure the router interfaces connected to these links.

2.2. Transforming nodes in routers to support routing protocols

Free Range Routing[4] (a.k.a. FRRouting or FRR), is a free and open source Internet routing protocol suite for Linux and Unix platforms. It implements BGP, OSPF, RIP, IS-IS,

[3] `https://learn.fabric-testbed.net/knowledge-base/portal-slice-builder-user-guide`

[4] `https://frrouting.org/`

```
from ipaddress import ip_address, IPv4Address, IPv6Address, IPv4Network, IPv6Network

try:
    num_subnets = 3
    route_link_addrs = []
    for i in range(num_subnets):
        route_link_subnet = IPv4Network(f'192.168.{i+1}.0/24')
        route_link_available_ips = list(route_link_subnet)[1:]
        route_link_addr1 = route_link_available_ips.pop(0)
        route_link_addr2 = route_link_available_ips.pop(0)
        route_link_addrs.append({route_link_addr1, route_link_addr2})

except Exception as e:
    print(f"Exception: {e}")
```

Figure 6. Route link subnets

PIM, LDP, BFD, Babel, PBR, OpenFabric and VRRP, with alpha support for EIGRP and NHRP. The FRR provides a flexible, scalable, and robust routing solution for large-scale networks. FRR has gained significant interest in recent years due to its ability to support a wide range of network topologies and its support for multiple routing protocols. It launched in April 2017, when Quagga[5] forked. For this work, FRRouting was used on its version (v. 7.5.1).

The allocated nodes that are so far Linux Debian with some network interfaces need to become routers to run legacy routing protocols. **Figure 8** shows the configuration in each node that uploads and configures the FRRouting routing software. This complex configuration is handled through a bash script `frr_config.sh` that will be available in this shared notebook. The script is executed by, first, uploading the script with the `node.upload_file()` FABLib method. Then the script is executed using the `node.execute()` FABLib method. Note that the script passes the OS interfaces names and configured IPs as arguments from the notebook to the script.

Configure the local nodes with addresses from the local subnet available address list. Add routes to the other local subnets via the local gateway. The example creates the group network address in `192.168.0.0` and assigns to the specific subnet address of each router.

Three ranges of subnets are created, using the Python `ipaddress` library, for each network established between the SmartNICs in the node configuration (**Figure 3**). Each node has two VLAN interfaces to which IPs will be assigned. This configuration will be performed through a bash script, which will be sent to each node and executed with the following parameters: interface 1 name and IP, interface 2 name and IP, subnet to limit the exchange of routes of OSPF and router-id.

```
# Delete slice by name
try:
    slice_name = 'FRRouting_Experiment'
    slice = fablib.get_slice(name=slice_name)
    slice.delete()
except Exception as e:
    print(f"Exception: {e}")
```

Figure 7. Delete slice by name

[5] `https://github.com/Quagga`

By default, the slice has a validity of 24 hours unless renewed. The **Figure 7** demonstrates how to delete a slice by name. It is an important practice to delete the slice after the experiments, in order to reduce the idle time of allocated resources.

```
# Configure FRRouting on each node
try:

    slice_name = 'Triangle_Topology'
    slice = fablib.get_slice(name=slice_name)

    node_threads = []
    subnets = [['net1', 'net3'], ['net2', 'net1'], ['net3', 'net2']]
    router_ids = ['1.1.1.1','2.2.2.2','3.3.3.3']
    nodes = slice.get_nodes()

    for i, node in enumerate(nodes):
        print(f'Config Router {i+1}')
        node.upload_file('./frr_config_rocky.sh','frr_config_rocky.sh')
        node_iface1 = node.get_interface(network_name=subnets[i][0])
        node_iface2 = node.get_interface(network_name=subnets[i][1])
        node_config_thread = node.execute_thread(f'chmod +x frr_config_rocky.sh && sudo ./frr_config_rocky.sh\
                                    {node_iface1.get_os_interface()} {route_link_addrs[i][0]}\
                                    {node_iface2.get_os_interface()} {route_link_addrs[i][1]}\
                                    192.168.0.0 {router_ids[i]}')
        node_threads.append(node_config_thread)

    print(f"Joining Threads")
    for i, thread in enumerate(node_threads):
        stdout, stderr = thread.result()
        print(f"Router {i+1}: ", stdout, stderr)

except Exception as e:
    print(f"Exception: {e}")
```

Figure 8. Upload and execute FRRouting config script

3. Running experiments in FABRIC testbed

The tests were conducted on the FABRIC platform using a Jupyter Notebook. The codes used in the tests can be accessed from the article's repository on GitHub[6]. Each node in the platform was equipped with 2 cores, 8 GB of RAM, and 10 GB of disk space. The operating system employed for the tests was Rocky Linux 8.5 (Green Obsidian), while the interfaces used were dedicated NICs "Mellanox ConnectX-5 Dual Port 10/25GbE". **iperf3**[7] (v. 3.5) was used to generate the traffic and measure the throughput.

3.1. OSPF: failure recover use case

In order to manage events in this experiment, such as injecting a fault in a network link, the FABlib API is used through the `execute_thread()` command of the `Node` class. This command enabled us to execute commands in parallel through the routers, and we were able to disconnect the `eth2.100` interface of the `R1` router while running the ping command to generate ICMP packets to be delivered to the `eth2.100` interface of router `R3` with IP `192.168.2.2`. More specifically, we triggered a link failure in 10 seconds, while traffic was generated, which caused the OSPF protocol to detect and recover from the failure by rerouting traffic through the remaining path.

As depicted in **Figure 9**, there was an abrupt increase in RTT due to the route update to the new path. It's worth noting that no packet loss occurred during this experiment,

[6] `https://github.com/edgardcunha/wtestbeds2023`
[7] `https://iperf.fr/`

though. Besides, the shortest path (link between routers `R1` and `R3`) was re-established at 20-seconds mark. As a result, after a few seconds at time 27 sec, RTT is reduced to the same value before the failure.

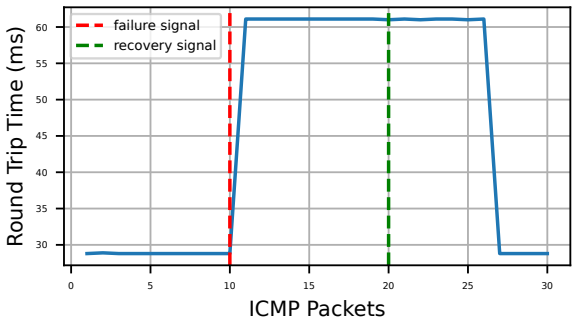

Figure 9. RTT with events of path failure and path recovery

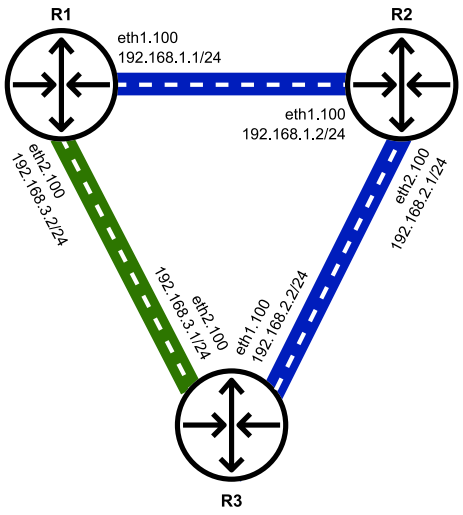

Figure 10. Topology and IP Addressing

3.2. Throughput use case

This experiment involves the evaluation of the maximum throughput between the routers network interfaces reaching the capacity of the dedicated smartNICs crossing the paths. A simple workload of UDP traffic is generated on the directly connected links using iperf3 for 30 seconds. The existing paths were explored (`R1 -> R2`, `R2 -> R3` and `R3 -> R1`) for each different traffic rate, which were: 1, 3, 5 and 10 Gbps. Different from the previous test, threads were not used for this evaluation, only the `execute()` command of the `Node` class of the FABlib API.

Table 1. Network Adapter Tuning Parameters

Flags	Values
`ipv4.tcp_mtu_probing`	1
`core.rmem_max`	2147483647
`core.wmem_max`	2147483647
`ipv4.tcp_rmem`	4096 87380 2147483647
`ipv4.tcp_wmem`	4096 65536 2147483647

Figure 11 shows the data obtained from a path `R1 -> R2`, for which the throughput achieves around 3 Gbps. All the three paths have been evaluated and essentially they presented similar performance results with small variations on throughput. Although the dedicated NIC model `NIC_ConnectX_5` indicates that its capacity is of 25 Gbps, by adjusting certain TCP/IP stack parameters for tuning the network adapter in the hosting OS, the throughput achieves a rate of near 10 Gbps (see **Figure 12**). The tuning parameters are described in **Table 1**, and basically set to limit to the maximum receive buffer size that applications can request (*proc/sys/net/core/rmem_max)*, and set maximum size of the

Memory Receive Buffer per connection (*/proc/sys/net/ipv4/tcp_rmem)*. They define the actual memory usage, not only TCP window size[8].

Note that this is a raw experiment with the purpose to explore the existing features of the testbed API with preliminary performance results. There is neither use of DPDK to accelerate packets forwarding nor the P4 offloading code at the smartNICs that could be used to increase the maximum throughput available in the network links.

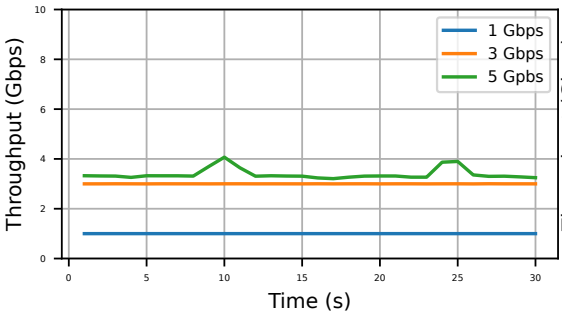

Figure 11. Default OS/NIC config

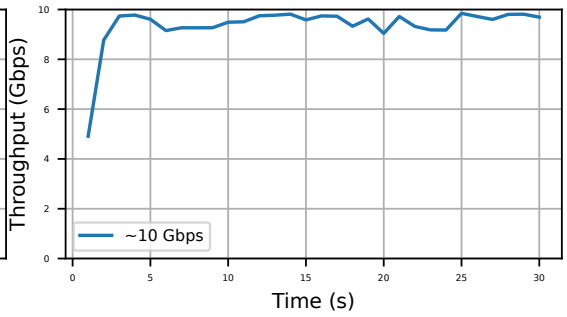

Figure 12. Tuning configuration

3.3. Lessons Learned

One of the main lessons learned from the FABRIC testbed was the use of Shared and Dedicated NICs and bindings to these interfaces, called Network Services. It was necessary to upgrade OS Debian 10 to Debian 11 for the 25 Gbps Dedicated NIC to work properly. It is important to point out that this error did not occur when using the OS Rocky Linux, for example, which was used in the two experiments presented above. Even with extensive documentation with several practical examples of using the testbed, errors were not mentioned when using the Debian OS 10 with Dedicated NICs. In this way, it is necessary to develop a list containing the results of tests of use and performance of the interfaces in each OS, with the objective of marking out subsequent tests.

It is noteworthy that when interacting on the platform forum, the FABRIC maintainers added Debian 11 to the default list of operating systems available for creating slices/nodes.

4. Conclusion

This work provided a short introduction to FABRIC testbed with arguments in favor of its use when creating open and reproducible experiments. A brief context of open testbeds for networking experimentation was presented. In order to disseminate FABRIC testbed usability to the research and education community, installation and entry-level configurations were presented.

However, FABRIC testbed has much to improve and welcomes the networking community worldwide to help developing: **i)** networking programming/configuration by examples; i.e. guidelines on how to use programmable dedicated smartNICs devices or even DPDK technology for packets forwarding acceleration **ii)** strategies to make it easy to use also for fostering innovation.

[8] `https://fasterdata.es.net/host-tuning/linux/udp-tuning/`

As future work, we envision to deploy PolKA source routing protocol [Dominicini et al. 2020] and its recent extension for multi-path [Guimarães et al. 2022], as a next step to advance our previous experience in protocol deployment at testbeds reported in [Borges et al. 2022a, Dominicini et al. 2021].

5. Acknowledgments

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

References

Baldin, I., Nikolich, A., Griffioen, J., Monga, I. I. S., Wang, K.-C., Lehman, T., and Ruth, P. (2019). Fabric: A national-scale programmable experimental network infrastructure. *IEEE Internet Computing*, 23(6):38–47.

Berman, M., Chase, J. S., Landweber, L., Nakao, A., Ott, M., Raychaudhuri, D., Ricci, R., and Seskar, I. (2014). Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61:5–23. Special issue on Future Internet Testbeds – Part I.

Borges, E., Pontes, E., Dominicini, C., Schwarz, M., Mate, C., Loui, F., Guimarães, R., Martinello, M., Villaça, R., and Ribeiro, M. (2022a). A lifecycle experience of polka: From prototyping to deployment at géant lab with rare/freertr. In *Anais do XIII Workshop de Pesquisa Experimental da Internet do Futuro*, pages 35–40, Porto Alegre, RS, Brasil. SBC.

Borges, E., Pontes, E., Mate, C., Loui, F., Martinello, M., and Ribeiro, M. (2022b). Freerouter in a nutshell: A protocoland routing platform for open and portable carrier-class testbeds. In *Anais do I Workshop de Testbeds*, pages 36–46, Porto Alegre, RS, Brasil. SBC.

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.

Dominicini, C. et al. (2020). Polka: Polynomial key-based architecture for source routing in network fabrics. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pages 326–334. IEEE.

Dominicini, C., Guimarães, R., Mafioletti, D., Martinello, M., Ribeiro, M. R. N., Villaça, R., Loui, F., Ortiz, J., Slyne, F., Ruffini, M., and Kenny, E. (2021). Deploying polka source routing in p4 switches : (invited paper). In *2021 International Conference on Optical Network Design and Modeling (ONDM)*, pages 1–3.

Guimarães, R. S., Dominicini, C., Martínez, V. M. G., Xavier, B. M., Mafioletti, D. R., Locateli, A. C., Villaca, R., Martinello, M., and Ribeiro, M. R. N. (2022). M-polka: Multipath polynomial key-based source routing for reliable communications. *IEEE Transactions on Network and Service Management*, pages 1–1.

Huang, T., Yu, F. R., Zhang, C., Liu, J., Zhang, J., and Liu, Y. (2017). A survey on large-scale software defined networking SDN testbeds: Approaches and challenges. *IEEE Communications Surveys Tutorials*.

M., S. et al. (2014). Design and implementation of the ofelia FP7 facility: The european openflow testbed. *Computer Network*, 61:132–150.

Salmito, T. et al. (2014). Fibre-an international testbed for future internet experimentation. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos-SBRC 2014*, pages p–969.

Zink, M., Irwin, D., Cecchet, E., Saplakoglu, H., Krieger, O., Herbordt, M., Daitzman, M., Desnoyers, P., Leeser, M., and Handagala, S. (2021). The open cloud testbed (oct): A platform for research into new cloud technologies. In *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*, pages 140–147.