C3-Edge – An Automated Mininet-Compatible SDN Testbed On Raspberry Pis and Nvidia Jetsons

Josef Hammer, Dragi Kimovski, Narges Mehran, Radu Prodan, and Hermann Hellwagner
Institute of Information Technology, Alpen-Adria-Universität Klagenfurt, Austria
{firstname}.{lastname}@aau.at

***Abstract*—The challenging demands for the next generation of the Internet of Things have led to a massive increase in edge computing and network virtualization technologies. While there is vast potential for research in these areas, managing complex adaptive infrastructure is difficult, and experiments with real hardware are tedious to set up. Furthermore, proposed solutions often require expensive hardware or labor-intensive procedures to replicate and build on these ideas. With our *C3-Edge* testbed, we address these challenges and propose a novel approach for automated edge testbed setup with a low-cost software-defined network and adaptive infrastructure configuration. We validated the efficiency of our approach on a real-world computing continuum infrastructure. The evaluation results confirm that our flexible approach is suitable for all but the most bandwidth-intensive applications.**

***Index Terms*—Edge Computing, MEC, Fog Computing, IoT, Software-Defined Networking, SDN, OVS, OpenFlow, Testbed, Raspberry Pi, Nvidia Jetson, Emulation, Virtualization**

I. Introduction

The ever-increasing processing and storage demand of the next generation of Internet of Things (IoT) applications led to an evolution in distributed computing and initiated the transition of the cloud services closer to the end users [1]. The trend was enabled by the introduction of novel computing paradigms, such as edge computing and network virtualization technologies, including software-defined networks (SDNs). Most modern applications benefit highly from the low-latency communication characteristics of edge computing and the flexibility of virtualized networks.

Unfortunately, it becomes increasingly difficult to efficiently manage complex adaptive infrastructures while providing the IoT applications with the lowest possible execution time [2], [3]. Thus, the transition of IoT applications from the cloud to the edge opened new critical research challenges, including increased heterogeneity of resources and network technologies. However, experiments with real hardware are tedious to set up, and proposed solutions often require expensive hardware or labor-intensive procedures to replicate and build on these ideas. We address these challenges by proposing a novel approach for edge testbed setup with a low-cost SDN network and adaptive infrastructure configuration.

First, our proposed solution utilizes inexpensive single-board computers to provide easily replicable performance results and high flexibility with the software setup (fig. 1). For example, we run a Kubernetes cluster on the testbed for experiments. Apart from the ubiquitous Raspberry Pis, we

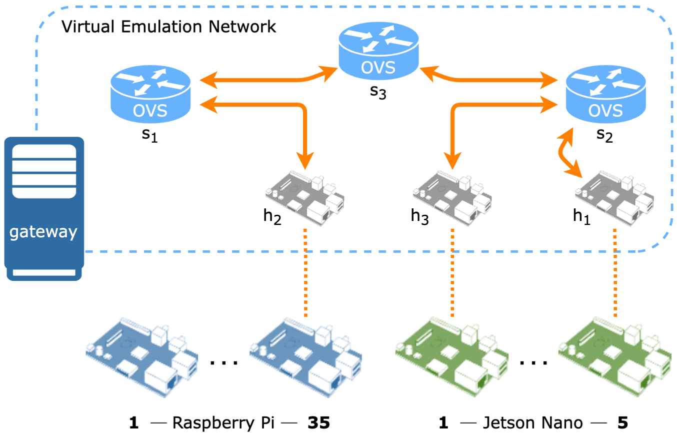

Fig. 1. High-level overview of the *C3-Edge* test infrastructure. As hosts (named h_i in the above emulation network topology), the virtualized emulation network uses a large number of Raspberry Pi and Nvidia Jetson Nano devices. Depending on the desired network topology, the hosts connect via multiple virtualized OpenFlow switches (s_i) – the latter run on the gateway.

added a few Jetson Nanos for machine learning experiments utilizing the GPU (such as in [4]). An automated setup enables quick replication of our testbed.

Second, our solution provides an optional, VLAN-based emulation overlay network for software-defined networking experiments. Most low-cost SDN testbeds utilize USB-based network adapters. Our novel approach uses VLANs instead to (i) avoid attaching a large number of adapters and (ii) enable more than five ports per virtual (OVS) switch.

Third, as an alternative to our custom configuration, our solution can parse the definition files of existing Mininet [5] network topologies and use them for the emulation network definition.

The main contributions of this work are:

- automated setup of edge infrastructure over single-board computers, such as Raspberry Pis and Jetson Nanos;
- a low-cost and flexible overlay network with automated setup for software-defined networking experiments;
- both large-scale and portable edge testbed systems for different usage applications;
- an empirical analysis of the network performance of the deployed edge testbed; and
- we open-sourced *C3-Edge* under the liberal *MIT license* on GitHub [6].

II. Related Work

To begin, we address the work by Lantz et al. [5] for setting up an SDN test environment – the well-known Mininet. It allows rapid prototyping of SDN solutions on a single machine. However, since all the virtualized hosts run on a single physical host, a distinct software setup for the emulated hosts becomes challenging. This shortcoming led to the development of Containernet by Peuster et al. [7]–[9], which allows using containers and Virtual Machines as hosts. Nevertheless, getting precise performance values from these virtualized hosts is difficult since all hosts are running on a single machine, sharing their processing power. Furthermore, for edge/fog computing and IoT experiments, we require direct support for interfacing external sensors and actuators.

Due to their low cost per unit, Raspberry Pi devices are popular among researchers. Kim et al. [10], Han et al. [11], Ariman et al. [12], and most recently Toosi et al. [13] proposed testbeds for SDN research built on Raspberry Pis as OpenFlow [14] switches. Out-of-the-box, the Raspberry Pi possesses only a single Ethernet interface. All these approaches add additional USB-based network adapters to increase the number of network interfaces. Then Open vSwitch [15] is run on the Raspberry Pis to enable software-based switching between these interfaces. See section III for the disadvantages.

The work by Rainer et al. [16] does not add additional network interfaces and bases their *Named Data Networking (NDN)* testbed on Banana Pi switches, which come with five native Ethernet interfaces out-of-the-box. However, Banana Pis are less popular than Raspberry Pis and thus less readily available and with less (community) support. Another affordable option is the Zodiac FX [17]–[20]. The Zodiac FX is an OpenFlow SDN hardware switch comprising four Ethernet ports. However, all ports are limited to 100 Mbps.

Sørensen et al. [21] described all the steps for setting up a Raspberry Pi-based testbed for network coding experiments. Their approach is to set up and configure the SD card image file and flash the final image for each node. For future updates, however, flashing all the SD cards again from a modified image file is a manual, labor-intensive task. Thus, our approach is to deploy only an original base image to each node and remotely customize all nodes fully automatedly. If it should be necessary to switch to a new clean base image provided by a vendor, it is easier with our approach since all the customizations are automated. More importantly, it makes it easy to quickly apply changes and updates without physically accessing the testbed. Furthermore, we may use the same mechanism to set up the devices for specific experiments.

Bellavista and Zanni [22] already showed the feasibility of a containerized approach on Raspberry Pis, and Fahs and Pierre [23] ran Kubernetes on these small devices. Most recently, Mahmud and Toosi [24] proposed a framework for a distributed container-based edge and fog computing environment on Raspberry Pis. These and other publications [25], [26] show that tiny, affordable computers like a Raspberry Pi are capable enough for real-world use cases.

III. Architecture

A. Physical Network

As presented in [1], our *Carinthian Computing Continuum (C^3)* [27] testbed contains cloud, fog, and edge resources. In the remainder of this paper, we usually refer only to the edge computing layer (*C3-Edge*) when talking about "the testbed".

Fig. 2 shows the detailed architecture of the testbed infrastructure. Each device is connected with a single CAT6A cable.

In some instances, having a "private" test system can be beneficial, e.g., to test new settings without interfering with the colleagues' work. For such cases, we also developed a smaller version of the *C3-Edge* testbed – the *Portable Test System* [28]. One can easily transport it between different workplaces and connect to it locally, either by cable or wirelessly.

B. Virtual Emulation Network

For many experiments, the testbed with its physical network is already sufficient. For example, for [4], we run a Kubernetes cluster on the testbed. However, for our research [29], [30], we wanted a flexible, programmable emulation network based on the Raspberry Pis as hosts. Thus, we decided to design an overlay network on top of the physical network presented in the previous section. The overlay network should be able to form any desired topology and require as little hardware as possible. As we noted in section II, the usual approach to set up a software-defined network based on Raspberry Pis is to add USB-Ethernet adapters to add additional network interfaces. This approach has several disadvantages, however. First, it comes at additional costs for the Ethernet adapters and complicates the hardware setup. Second, such a switch can only have up to five ports without using an external USB hub. Furthermore, unlike the physical Ethernet interface on the Raspberry Pi 4B, only USB-based Ethernet interfaces on the two USB 3.0 ports could operate at 1 Gbps, while the two USB 2.0 ports share a single USB 2.0 hub internally [31].

With the goal of a more flexible network setup, we propose a different solution. Instead of using a fraction of our nodes as switches, we run all instances of OVS on the gateway machine. As a result, our emulation utilizes a virtually unlimited number of switches with a vast number of ports. To force all physical nodes to connect via the virtual switches instead of communicating directly with their peers, we use multiple VLANs, as shown in fig. 2. All nodes participate in the default (untagged) VLAN 1 that forms the management network. In addition, for the emulation network, each node is part of another (tagged) VLAN that contains only the node itself and the gateway.

While these tiny VLANs do not make much sense by themselves, they enable us to add specific nodes to our virtual (OVS) switches. Therefore, as presented in fig. 2, we can group arbitrary nodes to form a virtual network. As soon as we point the participating nodes' default route to the emulation network, this setup forces the nodes to communicate via the virtual switches. As an additional measure, we block outgoing traffic from participating nodes via the management network whenever an emulation network is active. Incoming traffic needs to stay allowed, though, to allow reverting the settings.

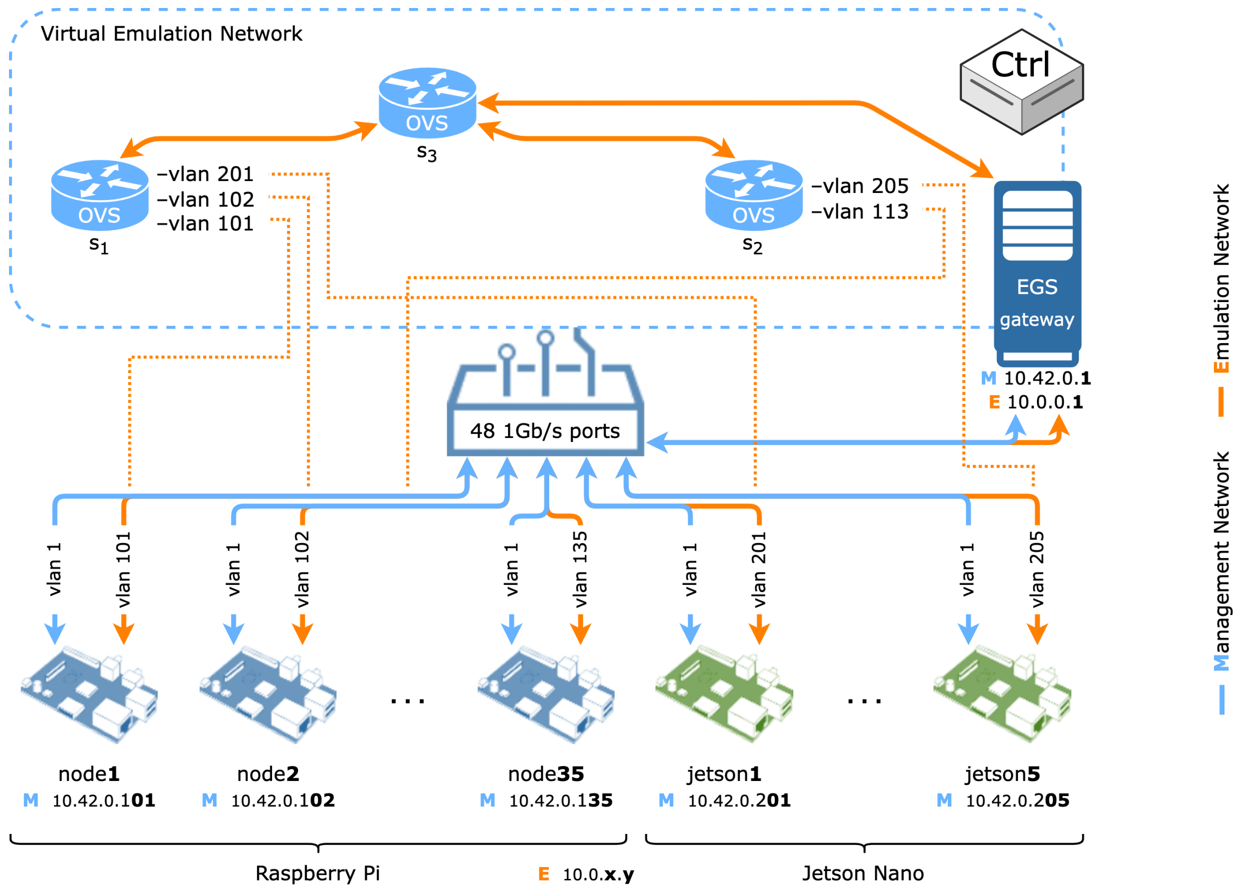

Fig. 2. The architecture of our test infrastructure. For the optional virtual emulation networks, each node has its separate VLAN. Depending on the desired emulation network topology, the VLANs are connected to multiple OpenFlow switches (s_i; running on the gateway). Whenever an emulation network is active, outgoing traffic from participating nodes via the management network is blocked, forcing the nodes to communicate via the virtual switches only.

IV. Setting up the Test Infrastructure

As discussed in section II, several publications have already described setting up a Raspberry Pi-based test system. However, one essential issue is the unnecessarily large number of manual steps required. While we see an immense value in the various steps and the reasoning behind them well explained, we believe in the core values of the *Agile Manifesto* [32], one of them being "*working software over comprehensive documentation*". Thus, when setting up the test infrastructure, our goal was to minimize the number of steps needed. See [6] for instructions on how to set up the testbed.

Once the testbed is set up, we can run many different emulation network topologies on top of it. Setting up a new emulation network topology is a two-step process: (i) defining the topology specification and (ii) activating the topology. See [30] for a simple example topology. The fastest and most effortless way to start is with an existing Mininet definition of the emulation network topology. MiniEdit comes with Mininet and is used for visually defining an SDN network topology, which it saves as a *MiniEdit Topology* (`*.mn`) file. We can use this file for the setup.

If an existing Mininet definition is unavailable, the second option to define a topology is a *YAML* [33] file with a *C3-Edge emulation definition*. The command `c3 [--clean] topo1` sets/cleans up the emulation network defined in `[topo1/]topo1.(yml|mn)`. See our repository [6] for topology definition examples and more details. If one decides to use the YAML file to define the emulation network topology but later would like to use that topology with Mininet, we provide a converter. The converter will take the YAML file and produce both a MiniEdit topology file and a Mininet script.

V. Evaluation

To evaluate the performance of the test infrastructure, we created two evaluation scenarios encompassing (i) direct connections and (ii) connections via the emulation network. For the first evaluation scenario, we involve two nodes and the hardware switch. In the latter scenario, all requests must go from the node via the hardware switch to the gateway and via the virtual OVS switch to the destination node. Unless the destination node is the gateway, this means going back from the gateway via the hardware switch to the node. As a result, the connection between the switch and the gateway becomes a potential bottleneck of the emulation system. All experiments were performed using Raspberry Pis (*RPi*) as nodes.

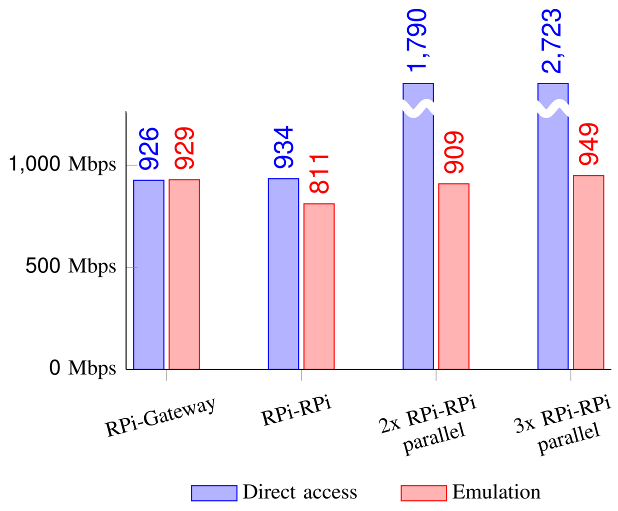

Fig. 3. Total throughput in Mbps between a Raspberry Pi and the EGS gateway or another Raspberry Pi. For the parallel tests, we ran `iperf3` on two and three independent pairs of Raspberry Pis simultaneously.

First, we measured the effective throughput of the system using the tool *iPerf3* [34] with its default settings. Fig. 3 presents the results of four different setups, each one using both direct access and the virtual emulation network. The first two setups test node-to-gateway communication (*RPi-Gateway*) and node-to-node communication (*RPi-RPi*). The latter two measure the total throughput with either two (*2x parallel*) or three (*3x parallel*) simultaneous node-to-node communications. As expected, we see no difference for node-to-gateway connections. However, we see a significant drop in throughput for the node-to-node connections. This drop is due to the overhead of routing every packet via the virtual switch on the gateway. The experiments with parallel connections show a much more significant difference. For these parallel tests, we avoid overlap between the nodes, i.e., we involve four and six RPis, respectively. Thus, with the direct connections, the only limitation is the hardware switch. As a result, the measured throughput almost doubles/triples. With the emulation, on the other hand, the upper limit is the bandwidth of the connection to the gateway. This bandwidth is shared equally between the two/three pairs. On a positive note, we see the total utilization of the connection increasing and converging to the link's maximum capacity. Furthermore, keep in mind that for realistic emulation networks, we frequently limit the bandwidth of links between nodes to lower values.

The latency, on the other hand, is not impacted by the bandwidth of the single connection. However, the routing via the gateway does increase the average round-trip time by 60 % (fig. 4). Nevertheless, the average latency we measured was still below one-third of a millisecond.

VI. Discussion

Our test system is highly flexible and allows quickly setting up any network topology. The usual approaches, discussed in section II, require multiple devices operating as OpenFlow-capable switches – either running OVS or a dedicated hardware switch. In contrast, our system allows running all virtual switches on a single gateway node. Furthermore, the single network cable to each node vastly simplifies the hardware setup and lowers the total cost of the test system.

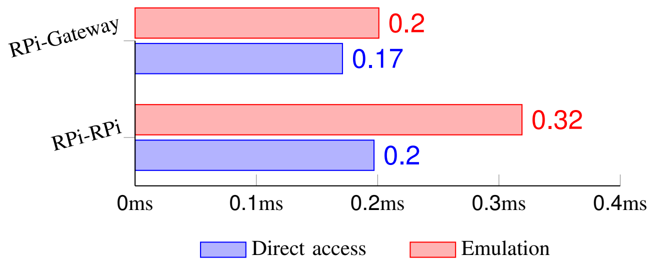

Fig. 4. Average round-trip time in ms between a Raspberry Pi and the EGS gateway or another Raspberry Pi. Measured with `ping -c 20`.

These advantages come at a price, however. With the current setup, all traffic needs to go through the gateway. As a result, the network connection between the switch and the gateway becomes a potential bottleneck of the system. Depending on the link's capacity between the switch and the gateway, this potential bottleneck might be relevant for use cases where many nodes require high bandwidth simultaneously. If, on the other hand, the experiments are focused on low latency and routing questions or have few nodes transmitting simultaneously, this convenient setup should be satisfactory.

There are several ways to mitigate this limitation. One is adding further network interface cards to the gateway and bonding several links to the switch to multiply the bandwidth. Another option would be to use one or more 10+Gbps ports on a switch (e.g., SFP+ ports) to connect to the gateway. However, these two options would come at additional costs. One of the free options would be to set up some of the virtual switches not on the gateway but on nodes instead. Furthermore, nodes could be allowed to communicate with each other directly when no OpenFlow control over their traffic is required (by adding the respective VLAN interfaces to each node). An example would be combining nodes to form a Kubernetes cluster.

VII. Conclusion

We introduced a novel automated approach for deploying both a large-scale and a portable edge testbed using SDN virtualization over Raspberry Pi and Nvidia Jetson Nano devices. The evaluation results confirm that our flexible approach is suitable for all but the most bandwidth-intensive applications. Due to the low costs and the high degree of automation, our solution enables easy *repeatability* and *replicability* of single-board computer and software-defined networking experiments. *C3-Edge* is available on GitHub (liberal *MIT license*) [6].

Acknowledgement

This work received funding from the Austrian Research Promotion Agency (FFG; Grant 888098, Kärntner Fog).

REFERENCES

[1] D. Kimovski, R. Matha, J. Hammer, N. Mehran, H. Hellwagner, and R. Prodan, "Cloud, Fog, or Edge: Where to Compute?" *IEEE Internet Comput.*, vol. 25, no. 4, pp. 30–36, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9321525/

[2] A. Ali-Eldin, B. Wang, and P. Shenoy, "The Hidden Cost of the Edge: A Performance Comparison of Edge and Cloud Latencies," in *Proc. Int. Conf. High Perform. Comput. Networking, Storage Anal.* ACM, 2021.

[3] D. Kimovski, N. Mehran, C. E. Kerth, and R. Prodan, "Mobility-Aware IoT Applications Placement in the Cloud Edge Continuum," *IEEE Trans. Serv. Comput.*, 2021.

[4] N. Mehran, Z. N. Samani, D. Kimovski, and R. Prodan, "Matching-based Scheduling of Asynchronous Data Processing Workflows on the Computing Continuum," *IEEE Int. Conf. Clust. Comput. ICCC*, 2022.

[5] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," *Proc. Ninth ACM SIGCOMM Work. Hot Top. Networks - Hotnets '10*, 2010.

[6] "C3-Edge: Carinthian Computing Continuum (C³) Edge Testbed." [Online]. Available: https://github.com/josefhammer/c3-edge.git

[7] M. Peuster, J. Kampmeyer, and H. Karl, "Containernet 2.0: A Rapid Prototyping Platform for Hybrid Service Function Chains," *4th IEEE Conf. Netw. Softwarization Work. NetSoft 2018*, pp. 335–337, 2018.

[8] M. Peuster, H. Karl, and S. Van Rossem, "MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments," *2016 IEEE Conf. Netw. Funct. Virtualization Softw. Defin. Networks, NFV-SDN 2016*, vol. 1, pp. 148–153, 2016.

[9] M. Peuster and H. Karl, "Profile your chains, not functions: Automated network service profiling in DevOps environments," in *2017 IEEE Conf. Netw. Funct. Virtualization Softw. Defin. Networks*. IEEE, nov 2017.

[10] H. Kim, J. Kim, and Y. B. Ko, "Developing a Cost-Effective OpenFlow Testbed for Small-Scale Software Defined Networking," *Int. Conf. Adv. Commun. Technol. ICACT*, pp. 758–761, 2014.

[11] S. Han and S. Lee, "Implementing SDN and Network-Hypervisor based Programmable Network using Pi Stack Switch," *Int. Conf. ICT Converg. 2015 Innov. Towar. IoT, 5G, Smart Media Era, ICTC 2015*, 2015.

[12] M. Ariman, G. Secinti, M. Erel, and B. Canberk, "Software Defined Wireless Network Testbed using Raspberry Pi OF Switches with Routing Add-on," *2015 IEEE Conf. Netw. Funct. Virtualization Softw. Defin. Network, NFV-SDN 2015*, pp. 20–21, 2015.

[13] A. N. Toosi, J. Son, and R. Buyya, "CLOUDS-Pi: A Low-Cost Raspberry-Pi based Micro Data Center for Software-Defined Cloud Computing," *IEEE Cloud Comput.*, vol. 5, no. 5, pp. 81–91, 2018.

[14] Open Networking Foundation, "OpenFlow Switch Specification v1.5.1," Open Networking Foundation, Tech. Rep., 2015.

[15] "Open vSwitch – Production Quality, Multilayer Open Virtual Switch." [Online]. Available: https://www.openvswitch.org/

[16] B. Rainer, D. Posch, A. Leibetseder, S. Theuermann, and H. Hellwagner, "A low-cost NDN testbed on banana pi routers," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 105–111, 2016.

[17] P. Zanna, P. Radcliffe, and K. G. Chavez, "A Method for Comparing OpenFlow and P4," *2019 29th Int. Telecommun. Networks Appl. Conf. ITNAC 2019*, pp. 22–24, 2019.

[18] A. Van Bemten, N. Derić, J. Zerwas, A. Blenk, S. Schmid, and W. Kellerer, "Loko: Predictable Latency in Small Networks," *Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol.*, 2019.

[19] J. Alcorn, S. Melton, and C. E. Chow, "SDN On-The-Go (OTG) Physical Testbed," *2017 IEEE Conf. Dependable Secur. Comput.*, 2017.

[20] S. Wang, K. G. Chavez, S. Kandeepan, and P. Zanna, "The Smallest Software Defined Network Testbed in the World: Performance and Security," *IEEE/IFIP Netw. Oper. Manag. Symp. Cogn. Manag. a Cyber World, NOMS 2018*, pp. 1–2, 2018.

[21] C. Sørensen, N. Hernández Marcano, J. Cabrera Guerrero, S. Wunderlich, D. Lucani, and F. Fitzek, "Easy as Pi: A Network Coding Raspberry Pi Testbed," *Electronics*, vol. 5, no. 4, p. 67, 2016.

[22] P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on docker containerization over RaspberryPi," *ACM Int. Conf. Proceeding Ser.*, 2017.

[23] A. J. Fahs and G. Pierre, "Proximity-Aware Traffic Routing in Distributed Fog Computing Platforms," *CCGrid*, 2019.

[24] R. Mahmud and A. N. Toosi, "Con-Pi: A Distributed Container-Based Edge and Fog Computing Framework," *IEEE Internet of Things J.*, vol. 9, no. 6, pp. 4125–4138, 2022.

[25] J. Islam, E. Harjula, T. Kumar, P. Karhula, and M. Ylianttila, "Docker Enabled Virtualized Nanoservices for Local IoT Edge Networks," *2019 IEEE Conf. Stand. Commun. Networking, CSCN 2019*, pp. 1–7, 2019.

[26] R. C. Kurniawan, R. Tulloh, and I. D. Irawati, "VPLS on Software Defined Network Using ONOS Controller Based on Raspberry-Pi 3," *Proc. - 2021 IEEE Asia Pacific Conf. Wirel. Mobile, APWiMob*, 2021.

[27] "The Carinthian Computing Continuum (C³)." [Online]. Available: https://c3.itec.aau.at/

[28] "C3-Edge – Portable Test System." [Online]. Available: https://josefhammer.com/r/PortableTestSystem/

[29] J. Hammer, P. Moll, and H. Hellwagner, "Transparent Access to 5G Edge Computing Services," in *2019 IEEE Int. Parallel Distrib. Process. Symp. Work.* IEEE, 2019, pp. 895–898. [Online]. Available: https://ieeexplore.ieee.org/document/8778343/

[30] J. Hammer and H. Hellwagner, "Efficient Transparent Access to 5G Edge Services," in *2022 IEEE 8th Int. Conf. Netw. Softwarization*. IEEE, 2022, pp. 91–96. [Online]. Available: https://ieeexplore.ieee.org/document/9844066

[31] "Raspberry Pi 4 USB Port Documentation." [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#raspberry-pi-4 (accessed 2023-01-29).

[32] "Agile Manifesto – Manifesto for Agile Software Development." [Online]. Available: https://agilemanifesto.org/

[33] "YAML Ain't Markup Language - The YAML File Format." [Online]. Available: https://yaml.org/

[34] "iPerf - The ultimate speed test tool for TCP, UDP and SCTP." [Online]. Available: https://iperf.fr/