

Ygarapé: Emulated Transport Networks for 5G Testbeds using Mininet in Kubernetes

Kauan Miranda, Ana Clara Dantas, Cleverson Nahum, Lucas Costa, João Fleury, Mariana Gonçalves, Cristiano Bonato, Glauco Gonçalves, Aldebaro Klautau, Silvia Lins, Andrey Silva

Abstract—Fifth Generation (5G) Networks research requires high-fidelity testbeds that balance the scalability of emulation with the realism of bare-metal setups. This paper introduces Ygarapé, a cloud-native architecture solution for transport network emulation that integrates Mininet into Kubernetes. Using an automated *interface hijacking* and a Representational State Transfer (REST) API for real-time control, Ygarapé enables containerized 5G Network Functions (NFs) to interact via programmable emulated topologies. Experimental evaluations featuring Ultra-Reliable Low-Latency Communication (URLLC) for Cloud Gaming and Enhanced Mobile Broadband (eMBB) for Video Streaming traffic show that Ygarapé accurately captures application-level performance degradation under transport stress. Results demonstrate that a 100 ms link delay triggers immediate FPS drops and latency spikes, which are autonomously mitigated by the platform’s control plane via Service Level Agreement (SLA) enforcement and path re-provisioning. The result is a flexible, low-cost platform ideal for validating closed-loop automation in 5G environments.

Keywords—Testbed, Transport Networks, 5G Network Functions.

I. INTRODUCTION

The advent of the Fifth Generation (5G) mobile networks has introduced a paradigm shift in telecommunications, characterized by highly demanding use cases such as Enhanced Mobile Broadband (eMBB) and Ultra-Reliable Low-Latency Communication (URLLC) [1]. To support these diverse requirements, 5G architectures rely on advanced virtualization technologies, including Network Function Virtualization (NFV) and Software-Defined Networking (SDN) [2]. In this context, experimental testbed research is critical for product design and validating novel algorithms under realistic operational constraints that abstract models cannot fully capture.

While discrete-event simulators are frequently used for initial evaluations of 5G transport networks due to their low cost, they often fail to capture the nuanced characteristics and *fidelity* of real protocol interactions [3], [4]. This gap is particularly relevant for the transport network layer, where precise timing directly impacts end-to-end performance. Consequently, there is an increasing demand for *experimental*

platforms that can host actual software stacks, such as OpenAirInterface (OAI) and Mininet, while providing fine-grained control over the emulated transport underlay.

Most high-fidelity testbeds are either owned systems possessed by large network operators or require substantial financial investment in bare-metal hardware and robust physical infrastructures [5], [6]. Several related works have attempted to bridge this gap by developing specialized platforms for emulating transport networks. For instance, architectures integrating SDN controllers with optical transport networks [7] or full-scale Management and Orchestration (MANO) frameworks [2], [8] have been proposed to manage network slicing. However, these solutions often lack the flexibility required for rapid experimentation, as many are tied to bare-metal configurations or heavy, resource-intensive orchestration layers that slow operability and increase maintenance costs [9].

To address these limitations, we introduce the *KA6G Ygarapé*, a cloud-native architecture for high-fidelity transport network emulation within virtualized environments. By integrating Mininet into the Kubernetes ecosystem, Ygarapé enables researchers to expose transport behavior as a controllable experimental variable [10]. This approach provides the agility of container orchestration without sacrificing the precision of kernel-level emulation, enabling the independent manipulation of functional paths (e.g., N2 and N3 interfaces) to mimic a physical switch fabric without the associated hardware overhead.

The main contributions of this work are: (i) a Cloud-Native transport emulation framework, consisting of a virtualized architecture that encapsulates network emulation as a manageable service for 5G experimentation, inheriting modern observability and lifecycle management; (ii) a Kubernetes-Native integration providing full support for cluster deployments, leveraging Kube-OVN for control-plane transparency and a specialized Multus/macvlan setup for data-plane isolation; (iii) a High-Fidelity traffic steering methodology that transparently routes 5G control and user plane traffic through programmable topologies without requiring dedicated hardware; and (iv) a Programmable Runtime Control Plane featuring an automated interface for real-time manipulation of network characteristics during active experiments.

II. YGARAPÉ: A PROGRAMMABLE CONTROL PLANE ARCHITECTURE FOR 5G TRANSPORT EMULATION

In this section, we present Ygarapé Mininet, a solution for integrating Mininet within Kubernetes-based Testbeds for

Kauan Miranda, Ana Clara Dantas, Lucas Costa, Cleverson Nahum, João Fleury, Mariana Gonçalves, Glauco Gonçalves and Aldebaro Klautau are with Universidade Federal do Pará (UFPA), Belém, Brazil. Cristiano Bonato is with Universidade do Vale do Rio dos Sinos (UNISINOS). Silvia Lins and Andrey Silva are with Ericsson Research, Brazil. Emails: [kauan.tavares@itec.ufpa.br, ana.bahia@ufpa.edu.br, cleversonahum@ufpa.br, lucas.queiroz.costa@itec.ufpa.br, joao.fleury@itec.ufpa.br, mariana.ribeiro@itec.ufpa.br, cbboth@unisinos.br, glaucogoncalves@ufpa.br, aldebaro@ufpa.br, silvia.lins@ericsson.com, andrey.silva@ericsson.com].

Transport Network Emulation. Ygarapé extends Mininet's functionality to provide a programmable control plane for building network topologies and robust interaction during execution. It comprises a REST API that interfaces real-time requests for changes to a central executor, called Engine, which stores the network state in a graph structure containing hosts, switches, links, and the necessary metadata to integrate with external tools while executing common network operations. The proposed methodology is organized into three main pillars: the cloud-native infrastructure that manages the lifecycle of 5G functions, the dual-network strategy that overcomes Kubernetes (K8s) transparency, and the Ygarapé engine that provides real-time programmatic control over the emulated transport underlay.

A. Hybrid Orchestration and Multi-Layer Infrastructure

The primary goal of the Ygarapé architecture is to bridge the gap between high-level cloud orchestration and low-level network fidelity. To ensure horizontal scalability and strict resource isolation, the environment is designed upon a hierarchical virtualization stack, as illustrated in Figure 1. The physical foundation consists of three servers interconnected by a dedicated L2 switch. Each physical host runs Proxmox to manage virtual machines that serve as K8s worker nodes. Within this setup, K8s acts as a lightweight Management and Orchestration (MANO) proxy, managing the components deployment and resource allocation via Helm Charts.

To prevent cross-domain bottlenecks, specific workers are dedicated to distinct functional domains: Worker 1 hosts the Radio Access Network (RAN) (Pod gNB), Worker 2 handles the emulated Transport (Ygarapé Pod), and Worker 3 hosts the Core Network (CN) functions. This distribution enables the platform to inherit a cloud-native architecture that ensures consistent, predictable performance through strict resource requests and limits. By isolating the Next Generation NodeB (gNB) and Core Network pods on dedicated hardware resources, the system ensures that performance variations result from intentional transport impairments rather than from infrastructure resource contention.

At the foundational level, Kube-OVN serves as the primary Container Network Interface (CNI), leveraging Generic Network Virtualization Encapsulation (GENEVE) tunnels [11] to provide a transparent K8s network. However, this tunneling mechanism abstracts the underlying topology, making it unsuitable for transport emulation that requires precise path control. To overcome this, we employ Multus CNI with the macvlan driver to establish an isolated data plane. This lightweight approach partitions the master interface (e.g., ens18) into virtual sub-interfaces with unique Media Access Control (MAC) addresses, granting direct physical-layer access necessary for the emulated transport underlay.

We implement a dual-network architecture in which Kube-OVN manages the control and management planes across virtual machines, while a parallel, isolated data plane is established using Multus and macvlan specifically for experimental 5G traffic. Multus assigns multiple secondary interfaces to the pods, which are pre-configured with static Internet Protocol

(IP) addresses via pod annotations. This separation ensures that experimental traffic is isolated from the K8s management traffic, enabling the creation of a dedicated, high-performance transport path between the RAN and the CN.

B. Virtual Network Hijacking

A critical aspect of this implementation is the duality of the Ygarapé Mininet pod environment. The main consequence of containerizing Mininet is the requirement for privileged execution to interact with the host's Linux kernel. The network objects it generates (i.e., hosts, switches, and links) are implemented as Linux network namespaces. Because this is the same fundamental technology used by K8s to achieve pod isolation, the engine process can operate at the same kernel level as the pods, enabling it to see both the pod's root namespace and the experimental namespaces simultaneously.

Within the container's view, the pod's network namespace is treated as the default root. This configuration allows the Mininet (Mininet) engine to locate the Multus interfaces (e.g., n2-amf, n3-gnb) that are attached to the Virtual Machine (VM)'s ens18 master interface. During topology initialization, the Ygarapé engine identifies the pre-configured secondary interfaces (assigned via Multus with static IP and MAC addresses in the pod annotations) and migrates them into specific host objects namespaces. In particular, we designate host objects as edge gateways due to their support for IP layer manipulation, enabling them to intermediate, on one side, pods and, on the other, the emulated switch mesh.

By migrating these interfaces into the Mininet hosts' namespaces, we bridge the gap between the virtualized K8s plane and the emulated transport plane. Likewise, this namespace duality ensures a clean life cycle: when the Mininet process terminates, the kernel's namespace management allows the engine to "return" the hijacked interfaces to the pod's root namespace, keeping the K8s networking state consistent across subsequent experimental cycles.

This migration process effectively interposes the emulated transport network between the RAN and the CN. Because the gNB and the core functions treat these hijacked interfaces as their primary data paths, all 5G traffic is forced to traverse the emulated transport network. This configuration enables the dynamic injection of link impairments, such as bandwidth throttling or latency spikes, without compromising the stability of the primary K8s cluster. Upon termination of the Mininet process, the interfaces are safely returned to the pod's namespace, preserving the integrity of the K8s networking lifecycle.

C. SDN Control and Forwarding Orchestration

To bridge the gap between topological abstraction and actual traffic steering, the architecture integrates the Open Network Operating System (ONOS) as a centralized SDN controller. While the Ygarapé Engine manages the lifecycle of the emulated nodes and the physical interface migration process, it offloads the complex forwarding logic to ONOS via the OpenFlow 1.3 protocol. Structurally, the controller is deployed as an external service within the cluster, reinforcing

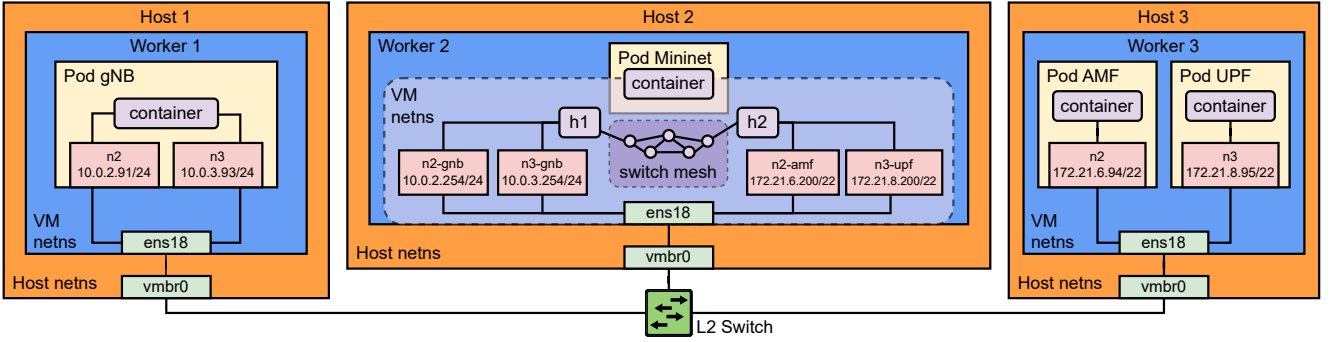


Fig. 1. Proposed architecture of the Ygarapé system. The diagram illustrates the distribution of 5G Network Functions (gNB, AMF, and UPF) across Kubernetes worker nodes, interconnected via a dedicated Mininet pod that emulates the transport network topology by intercepting N2 and N3 traffic interfaces.

a distributed, cloud-native design in which the control plane is decoupled from the emulation execution.

A dedicated integration layer, named ONOS Controller, implemented within the code base, enables the Engine to interact with ONOS's REST API. A critical technical distinction of this work is the proactive host registration mechanism: rather than waiting for the controller to learn node locations through passive packet-in events, the Engine immediately registers the Mininet's hosts used as edge access for the 5G network functions, such as the gNB and CN functions, into the ONOS Intent Framework.

The Mininet hosts function as gateways interfacing the emulated environment with K8s pods, enabling the orchestrator to compute deterministic, loop-free paths across the switch mesh using Dijkstra-based routing. By bridging these containerized functions with known MAC and IP bindings, researchers can implement sophisticated traffic engineering policies, such as explicitly separating Control Plane (CP) and User Plane (UP) flows. Consequently, the transport underlay is transformed from a static connectivity pipe into a programmable and observable component of the 5G ecosystem.

To facilitate high-performance interaction during active experiments, a REST interface using the FastAPI ecosystem runs as a daemon thread in parallel with the primary emulation process. This concurrent architecture allows the engine to interact directly with Mininet and Open vSwitch (OVS) objects within the same memory scope. As a result, researchers can issue PATCH requests to dynamically reconfigure the network. This capability transforms the emulation into a programmable control plane, allowing for the real-time evaluation of how different 5G functions respond to localized path impairments, such as injecting latency into a specific signaling path, without requiring a restart of the physical or virtual service stack.

III. EXPERIMENTAL SETUP AND RESULTS

The infrastructure establishes a hybrid Hardware-in-the-Loop (HiL) environment where real-world 5G functions interact with a virtualized transport underlay. The radio segment employs a USRP B210 Software Defined Radio (SDR) integrated via direct USB pass-through to the OAI gNB pod, ensuring low-latency signal processing compliant with 5G standards. In the User Equipment (UE) domain, experimental

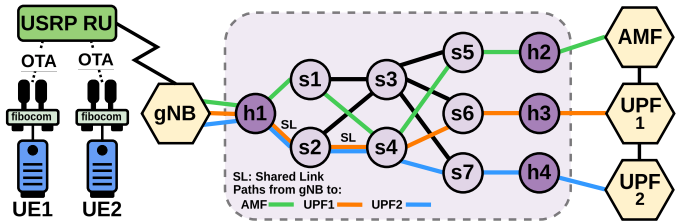


Fig. 2. Unified emulated transport topology. 5G control (green) and data (blue and orange) planes are segregated across the switches to evaluate the impact of dynamic QoS constraints on 5G core functions and radio units.

realism is further enhanced by utilizing physical Fibocom modems connected to dedicated workstations. Unlike pure software-based user simulations, this approach employs Over-the-Air (OTA) communications to link terminals to the gNB. The selection of real radio transmission aims to achieve a superior level of experimental fidelity by capturing the stochastic behavior of actual radio channels. These high-fidelity field data are essential for developing new algorithms, providing a rigorous validation environment for radio resource management and end-to-end performance optimization.

This unified configuration, illustrated in Fig. 2, employs a 7-switch topology that serves as the foundational baseline for all subsequent experimental scenarios. This setup provides a highly controlled validation field for evaluating transport-layer impacts on end-to-end performance without the need for redundant structural definitions.

A. Functional Path Isolation

The first experimental scenario evaluates the proposed architecture's capacity for granular, path-specific manipulation within the unified Hardware-in-the-Loop (HiL) environment. To demonstrate functional isolation within the 7-switch topology, we established segregated baseline routes: Control Plane (CP) traffic traversed $s_1 \rightarrow s_4 \rightarrow s_5$, while User Plane (UP) traffic utilized $s_2 \rightarrow s_4 \rightarrow s_6$. During concurrent operations, a UP Iperf3 stream and a CP monitoring flow, an automated script utilized the Ygarapé Application Programming Interface (API) to dynamically inject a 25 ms delay and 10% packet loss specifically into the CP endpoints (h_1-s_1 and s_5-s_4), leaving the UP path untouched.

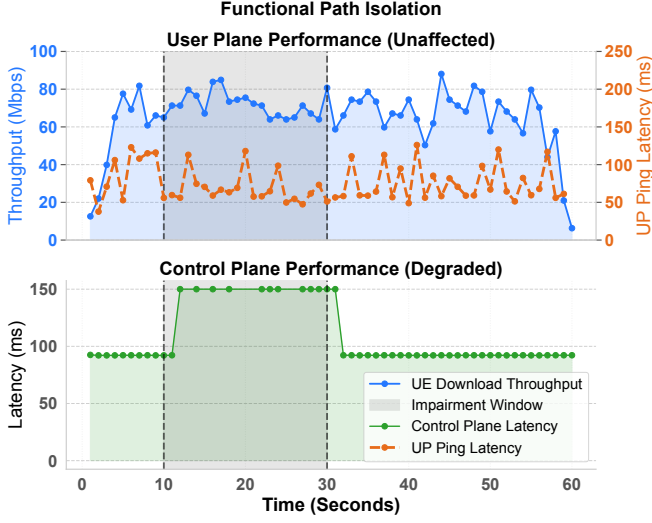


Fig. 3. Experimental results of functional path isolation. The impairment window highlights the degradation of the control-plane (N2) while the data-plane (N3) remains stable.

As shown in Fig. 3, the platform achieves strict functional isolation. Under normal conditions, CP RTTs maintain a stable baseline of approximately 92 ms. However, during the impairment window (10s to 30s), the CP latency accurately reflects the injected degradation, surging to nearly 145 ms. In contrast, the UP latency remained entirely unaffected by the transport manipulation. Anchored by the physical hardware, the UP RTT exhibited only the natural stochastic variations inherent to the OTA SDR link (37.7–165 ms), completely ignoring the deterministic delay spikes imposed on the CP. Crucially, the active data session never disconnected, proving the framework properly segregates traffic and reliably evaluates 5G signaling robustness without corrupting data-plane fidelity.

B. Path Isolation and Automated Re-routing for Slicing SLAs in Transport Network

To evaluate Ygarapé’s closed-loop orchestration, we implemented a hybrid End-to-End (E2E) slicing scenario using the OAI stack under a single Slice/Service Type. Service differentiation is achieved by deploying two distinct Data Network Names (DNNs) and independent User Plane Functions (UPFs), where each network slice is defined as a programmable transport path mapped to specific ONOS intent constraints. These Service Level Agreement (SLA) requirements, such as latency bounds for URLLC, are declared in the topology configuration and enforced dynamically by the SDN controller.

The emulated topology utilizes specific Mininet hosts as edge gateways to bridge the containerized components: h_1 (gNB Gateway) serves as the primary entry point for gNB traffic, utilizing a trunk link to support both control and data plane Virtual Local Area Networks (VLANs); h_2 (AMF Gateway) is dedicated to handling signaling traffic for the Access and Mobility Management Function; and h_3 (UPF1 Gateway) along with h_4 (UPF2 Gateway) act as edge gateways for two independent UPF instances, processing traffic for DNN 1 and DNN 2, respectively.

Service separation is enforced at the transport layer using VLAN tagging and intent-based forwarding policies. The URLLC slice is associated with VLAN 10, linking the h_1 and h_3 gateways through a high-priority intent with a strict $LatencyConstraint \leq 5.0$ ms. This slice supports a demanding cloud gaming application in which the UE 1 terminal uses the Moonlight client to stream the game Xonotic from a dedicated laboratory server. To guarantee this level of service, the traffic is steered through a prioritized path consisting of $h_1 \rightarrow s_2 \rightarrow s_4 \rightarrow s_6 \rightarrow h_3$. Simultaneously, the eMBB slice operates on VLAN 20, connecting h_1 to the h_4 gateway with a $BandwidthConstraint \geq 400.0$ Mbps focused on throughput stability. This slice hosts a video-streaming application on UE 2, which retrieves high-resolution content from a separate laboratory traffic server. This flow traverses a different transport route $h_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_7 \rightarrow h_4$. Ygarapé automatically translates these definitions into ONOS flow rules, ensuring the switching fabric adheres to the performance metrics specified in the topology configuration.

Ygarapé enables the rigorous evaluation of 5G SLA assurance and path separation by exposing the emulated transport underlay as a programmable experimental variable. To demonstrate this capability, we established a reactive closed-loop workflow utilizing an external orchestrator and a runtime degradation tool. The orchestrator is a standalone Python script that continuously monitors the network state and autonomously updates SDN intents via the ONOS Intent Framework to enforce service requirements. Complementing this, a degradation script acts as a user-defined tool that interacts with the Ygarapé REST API at runtime to simulate physical transport stress during active experiments.

The experimental execution follows a deterministic three-stage sequence:

- 1) Initialization: All containerized 5G services, including the Cloud Gaming (Xonotic/Moonlight) and Video Streaming applications, are initiated and stabilized.
- 2) Orchestration: The external orchestrator script begins its monitoring cycle, verifying that both slices are traversing their primary assigned paths.
- 3) Runtime Trigger: A degradation event is issued, injecting a synthetic 100 ms latency and bandwidth constraints specifically into the primary transport link (s_2-s_4). Once the orchestrator identifies the resulting SLA violation, it executes a routing update by re-provisioning the intents in ONOS, effectively rerouting the affected slice to a prioritized, non-congested transport route.

The experimental results, visualized in Figure 4, demonstrate Ygarapé’s capacity to support complex, closed-loop orchestration experiments. Initially, both slices share the s_2-s_4 link as their primary transport route. Upon the "Degradation Start" event, where latency is injected into this link, the external orchestrator detects the SLA violation for the URLLC slice. To mitigate the impact, the orchestrator re-provisions the ONOS intent by adding an *ObstacleConstraint* on the s_2-s_4 link, forcing the Cloud Gaming traffic to handover to a prioritized path through $s_2 \rightarrow s_3 \rightarrow s_6$. In contrast, the eMBB slice remains on the s_2-s_4 path, as its SLA is bandwidth-bound

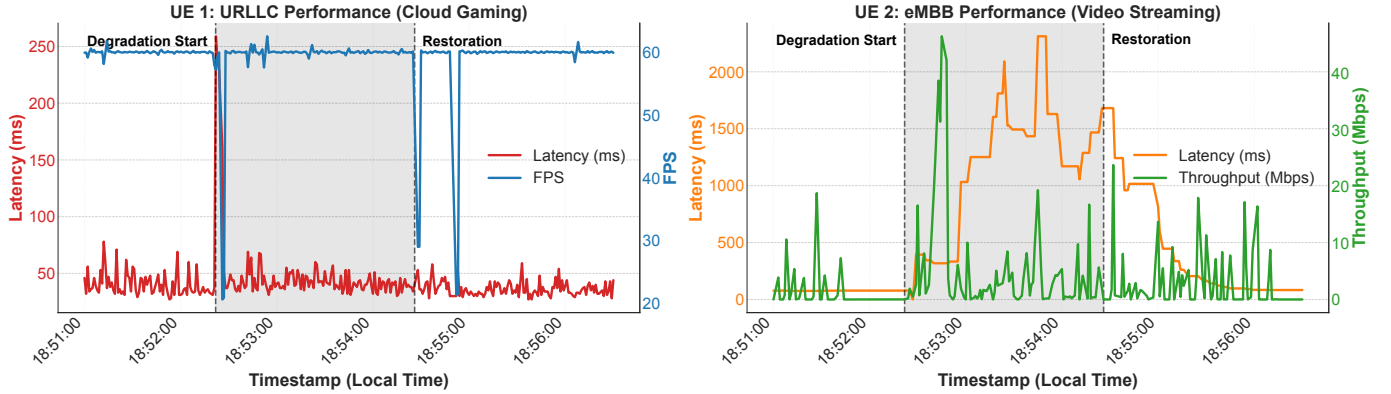


Fig. 4. Impact of transport-layer degradation on 5G user-plane metrics. The URLLC slice (left) demonstrates autonomous recovery through SDN-based path rerouting after an SLA violation is detected, while the eMBB slice (right) exhibits sustained latency spikes due to its fixed path assignment.

and does not trigger a latency-based reroute.

As shown in Figure 4, the Cloud Gaming application experiences a transient FPS drop and a latency spike immediately following the degradation. These artifacts represent the sum of the orchestrator’s detection time and the period required for ONOS to withdraw and reinstall the high-priority intent. Once the handover to the alternate path is complete, both FPS and latency return to nominal levels, ensuring service continuity despite the ongoing transport impairment.

Conversely, the Video Streaming application suffers from sustained User-Plane Quality of Experience (QoE) degradation throughout the impairment window, as it remains on the congested s_2 - s_4 link. Its RTT peaks above 2 seconds, illustrating the compounding effect of transport latency on the 5G user plane. Upon the "Restoration" of the s_2 - s_4 link, the orchestrator detects the improved conditions and removes the ObstacleConstraint, reverting the SLA slice to its primary path. This transition induces a second brief FPS drop, caused by the sub-second connectivity gap while ONOS updates the flow rules across the switch fabric. This sequence confirms that Ygarapé effectively enables the validation of advanced orchestration algorithms, such as SLA-aware path separation and dynamic rerouting.

IV. CONCLUSIONS

This paper presented Ygarapé, a cloud-native transport network emulation framework that bridges the gap between low-fidelity simulation and high-cost testbeds by integrating Mininet into the Kubernetes ecosystem. By leveraging a REST API and automated interface hijacking, Ygarapé effectively exposes transport behavior as a programmable experimental variable. Experimental evaluations with URLLC and eMBB slices confirmed the tool’s high fidelity, demonstrating how a 100 ms transport delay interacts with physical USRP/Fibocom hardware to amplify application-level RTT to over 2 seconds, while simultaneously enabling sub-second SLA recovery through autonomous SDN-based path rerouting. These results validate Ygarapé as a robust environment for closed-loop orchestration research, with future developments focused on P4 data plane integration and multi-cluster scalability.

ACKNOWLEDGMENT

This study was financed in part by Ericsson Telecomunicações Ltda., Brazil; the Brasil 6G project (01245.020548/2021-07); Project Smart 5G Core And MultiRAN Integration SAMURAI (MCTIC/CGI.br/FAPESP under Grant 2020/05127-2); OpenRAN Brazil - Phase 2 project (MCTI grant N° A01245.014203/2021-14); and partially supported by the National Institute of Science and Technology (INCT) STREAM, funded by the Brazilian National Council for Scientific and Technological Development (CNPq), process no. 409179/2024-8.

REFERENCES

- [1] J. G. Andrews *et al.*, "What will 5G be?" *IEEE Journal on selected areas in communications*, vol. 32, no. 6, pp. 1065–1082, 2014.
- [2] I. Afolabi *et al.*, "Network slicing and softwareization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.
- [3] K. Tan *et al.*, "Comparing simulation tools and experimental testbeds for wireless mesh networks," in *IEEE INFOCOM*, 2011, pp. 1–5.
- [4] L. Wischhof *et al.*, "Simulation vs. testbed: Small scale experimental validation of an open-source LTE-A model," in *IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*. IEEE, 2020, pp. 1–7.
- [5] T. Melodia *et al.*, "Colosseum, the world’s largest wireless network emulator," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 860–861.
- [6] A. Esmaily and K. Kralevska, "Small-Scale 5G Testbeds for Network Slicing Deployment: A Systematic Review," *Wireless Communications and Mobile Computing*, vol. 2021, no. 1, p. 6655216, 2021.
- [7] R. Muñoz *et al.*, "The ADRENALINE testbed: An SDN/NFV packet/optical transport network and edge/core cloud platform," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 116–123, 2017.
- [8] S. Fichera *et al.*, "On experimenting 5G: Testbed set-up for SDN orchestration across network cloud and IoT domains," in *IEEE Conference on Network Softwareization (NetSoft)*, 2017, pp. 1–5.
- [9] G. M. Yilma *et al.*, "Benchmarking open source NFV MANO systems: OSM and ONAP," in *6th IEEE Conference on Network Softwareization (NetSoft)*, 2020, pp. 232–234.
- [10] O. Arouk and N. Nikaein, "Kube5G: A cloud-native 5G service platform," in *IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.
- [11] J. Gross, I. Ganga, and T. Sridhar, "Geneve: Generic Network Virtualization Encapsulation," RFC 8926, Nov. 2020. [Online]. Available: <https://www.rfc-editor.org/info/rfc8926>