

On the Accuracy of Mininet: Comparing Network Emulation and Physical Testbeds for Distributed Systems

Luca Borgianni¹, Cristian Bua¹, Edoardo Coli², Davide Adami², Stefano Giordano¹

¹University of Pisa, Dept. of Information Engineering, Via G. Caruso 16, 56122 Pisa, Italy

²CNIT - Research Unit CNIT, Via G. Caruso 16, 56122 Pisa, Italy

Abstract—The increasing complexity of modern distributed systems, spanning cloud computing, edge devices, and ultra-low latency applications, needs accurate and scalable network emulation. This comparative study analyzes the fidelity of one of the most popular network emulators (Mininet) against a physical testbed to quantify performance deviations with a focus on scalability. Using iPerf for benchmarking, we measure bandwidth allocation dynamics under incremental client loads (1–20 nodes). Results show that Mininet accurately emulates throughput distribution across different nodes with errors below 9%. We further discuss Mininet’s limitations in replicating high-bandwidth links and propose a Python-based topology configuration tool to streamline emulation workflows. This work provides empirical insights into the fidelity and constraints of software-defined network emulation and the possibility of integration in a digital twin system.

Index Terms—Network Emulation, Mininet, Software-Defined Networking, Edge Computing, Distributed Systems

I. INTRODUCTION

The evolution of modern distributed systems, characterized by architectures such as cloud computing and edge computing, necessitates accurate and scalable network emulation. Particularly in the context of edge computing and distributed processing, the ability to realistically model network conditions becomes a key element in the evaluation and optimization of distributed infrastructures.

Network emulation plays a crucial role in the testing and validation of distributed systems, cloud-edge infrastructures, and next-generation Internet architectures. However, the fidelity and determinism of emulated environments remain key challenges, particularly under high concurrency. This paper investigates the performance of Mininet [1], a widely used network emulator, compared to a physical testbed under varying client loads. We evaluate a 21-node cluster comprising Rock4+ single-board computers and a TP-Link Gigabit switch, comparing it to a Mininet-emulated topology. Specifically, we focus on measuring the performance and throughput of the network under varying connection scenarios using *iPerf* [2], a widely adopted tool for network performance testing. Analyzing the results with a different number of nodes (1–20 nodes), we can assess the limitation and performance instability in network emulation, with its implications for large-scale network digital twins and

AI-driven network orchestration. Our findings suggest that while Mininet provides a cost-effective alternative, its reliance on CPU scheduling introduces non-deterministic behavior that must be accounted for in future network research and simulation frameworks.

Looking forward, the evolution of emulation platforms will play a foundational role in enabling large-scale network digital twins, where real-time feedback loops between physical and virtual infrastructures can drive autonomous decision-making. The integration of AI-driven orchestration, combined with advances in hardware acceleration and virtualization technologies, paves the way for emulation frameworks capable of supporting next-generation networks, such as 6G, quantum-assisted communication, and satellite-based edge infrastructures. To meet the demands of these futuristic scenarios, future emulators must evolve beyond current CPU-bound limitations, incorporating heterogeneous computing resources, predictive modeling, and self-adaptive simulation environments. These developments are not only necessary for improving fidelity and scalability, but also crucial to support the emergence of self-optimizing, resilient, and intelligent distributed systems.

The rest of the paper is organized as follows: Section II introduces previous work in this field, Section III presents the Methodology describing the Physical Testbed, the Emulated Testbed, and the process of Data Collection. Section IV presents the result, and finally, Section V concludes the work.

II. RELATED WORK

Several tools have been developed to provide researchers with flexible and cost-effective solutions for network emulation. Among these, Mininet, CORE, EstiNet, and other specialized emulators have gained significant attention.

Mininet is a widely used open-source network emulator designed for Software-Defined Networking (SDN) research [3]. It enables the creation of virtual networks on a single machine, allowing users to prototype and test SDN applications efficiently. However, Mininet is constrained by its single-machine execution model, which limits its scalability and performance [4]. To address these limitations, Mininet-HiFi was introduced, incorporating mechanisms to track and compensate for system

invariants such as queueing delay, transmission delay, and CPU capacity. Additionally, Containernet extends Mininet by adding support for Docker containers, enabling dynamic network modifications at runtime [5].

Common Open Research Emulator (CORE) is another popular network emulator that provides a flexible environment for modeling network topologies [6]. Its integration with EMANE allows for higher-fidelity link and physical layer modeling. Performance comparisons among different platforms using CORE have demonstrated that FreeBSD, leveraging the Netgraph subsystem, utilizes more CPU to process packets efficiently, whereas Linux-based implementations using Ethernet bridging exhibit comparable performance [7].

Another study focuses on comparing the scalability and performance of different network emulators. For instance, Mininet has been tested in large-scale network topologies ranging from 3 to 511 nodes, revealing execution times up to 242 seconds and memory usage reaching 700 MB. Despite its limitations in performance fidelity, Mininet remains a valuable tool for rapid prototyping and sharing experimental results [8]. Similarly, scalability tests conducted on various network emulators indicate that, apart from hardware processing power differences, performance trends remain consistent across multiple experiments [9].

A comparative study between EstiNet and Mininet evaluated their correctness, performance, and scalability using OpenFlow controllers. The study analyzed RTT measurements, failure rates in ping tests, and memory consumption. While EstiNet exhibited high accuracy and correctness, its simulation time increased significantly as the number of OpenFlow switches exceeded 961. Mininet, on the other hand, performed well but showed inconsistencies under specific network settings [10].

A crucial aspect of network emulation research is assessing how closely emulation results align with real-world network behavior. Some studies have explored this by comparing emulation results with real testbed experiments. For instance, measurements of throughput, latency, and file transfer times using *iPerf* and ping tests indicate that different platforms exhibit varying levels of performance fidelity. FreeBSD-based systems have demonstrated higher CPU utilization to enhance packet delivery, while Linux-based emulators offer stable but sometimes inconsistent results [11].

Other network emulators, such as NetEm and DummyNet, provide fine-grained control over network conditions. NetEm is integrated into the Linux kernel and enables the emulation of network impairments such as delay and packet loss, though it lacks built-in bandwidth control [12]. Conversely, DummyNet, primarily used in FreeBSD systems, allows precise bandwidth limitation, making it well-suited for testing applications under constrained network conditions [13].

To conclude, modern distributed systems rely on cluster networks to manage scalable workloads, necessitating robust tools for performance evaluation. Mininet enables cost-effective prototyping by virtualizing hosts, switches, and links. However,

its ability to replicate physical network behavior under high-throughput scenarios remains understudied.

III. METHODOLOGY

In this Section, we present the methodology of our test, describing the Physical Testbed and the Emulated Testbed. Moreover, we describe the data collection process.

A. Physical Testbed

The physical cluster consists of 21 nodes: one master and 20 Rock4+ clients interconnected via a TP-Link TL-SG1428PE switch, as shown in Fig. 1. Clients connect to the switch via 2 Gbps links (1 Gbps uplink/downlink).



Fig. 1. Physical Testbed Cluster

Let us give a detailed overview of the hardware specifications of the cluster, Rock4+ nodes, and the TP-Link switch:

Rock4+ Single-Board Computers:

- CPU: Rockchip RK3399 Hexa-core processor (Dual-core Cortex-A72 and Quad-core Cortex-A53)
- GPU: Mali-T860MP4
- RAM: 4 GB LPDDR4 dual channel
- Connectivity: Gigabit Ethernet (POE)
- Operating System: Linux-based distribution

TP-Link Switch (Model: TL-SG1428PE):

- Ports: 26x 10/100/1000 Mbps RJ45
- PoE Ports: 24x PoE (802.3at/af)
- Power Budget: Up to 350 W

The network topology of the cluster is presented in Fig. 2, in which one node is designated as the master (referred to as "Node0"), and the remaining nodes are connected using a switch, which can be described as a star topology. In this topology, the master node acts as the central hub or controller, while the switch serves as a central point for connecting all the other nodes. In the setup, the master node "Node0" is directly connected to the switch using a Gigabit Ethernet cable.

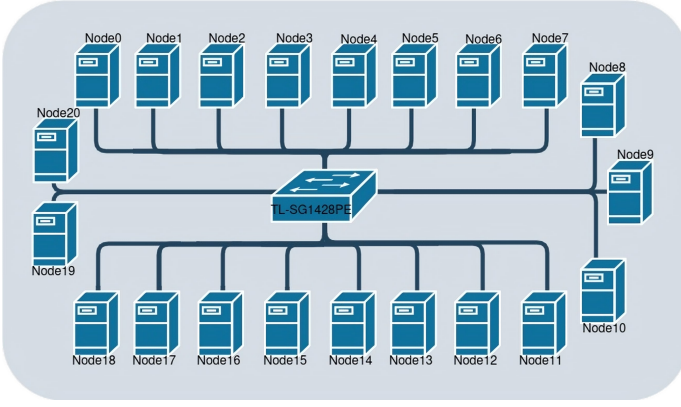


Fig. 2. Physical Cluster Topology

The switch acts as a central point of connectivity and provides multiple ports to accommodate the remaining 20 nodes. Each of the 21 nodes is connected to the switch using individual Gigabit Ethernet cables, with each cable connected to an available port on the switch providing a 2 Gbps bandwidth, 1 Gbps in UL, and 1 Gbps in DL.

B. Emulated Testbed

The Mininet topology replicates the physical setup using 21 hosts and a single OpenFlow [14] switch. This topology is instantiated using a custom-built topology generator that leverages the Mininet API within Python, enabling a declarative and flexible definition of network elements and their interconnections. The builder tool facilitates reproducibility and scalability via the configuration file, allowing for automated deployment of complex network scenarios. Due to Mininet's software switch limitations, the bandwidth of links between hosts and the switch is restricted to 1 Gbps.

Additionally, within the topology builder, the classic `mininet.cli` has been replaced with Python's multiprocessing functionality. With this change, different hosts in the Mininet network can operate concurrently, leveraging Python's multiprocessing to execute tasks independently without the constraints of the sequential Command Line Interface (CLI). By leveraging multiprocessing, the system improves parallelism, reduces command execution latency, and enhances the overall efficiency of the emulation, making it particularly well-suited for large-scale network simulations.

C. Data Collection

iPerf is an open-source tool used to measure and analyze network bandwidth and performance. It supports multiple transport protocols (TCP, UDP, SCTP) and provides advanced functionalities for testing throughput, latency, and jitter. *iPerf* operates based on a client-server model, where the client generates traffic towards the server and measures the connection's throughput. It can assess both the maximum available bandwidth and simulate traffic at controlled speeds to evaluate performance in real-world scenarios [15].

In our case, we used the TCP protocol to test network capacity. The client establishes a connection with the server on the specified port (default: 5201) and starts transmitting data. The generated TCP traffic follows the bulk data transfer model, meaning it sends the maximum amount of data possible while adhering to TCP protocol rules. TCP packets include:

- **TCP Header** (20 bytes)
- **IP Header** (20 bytes for IPv4, 40 bytes for IPv6)
- **Payload** (1448 bytes on an Ethernet network with an MTU of 1500)

TCP is a reliable and connection-oriented protocol, meaning it does not simply send data at the highest possible speed but follows key phases to optimize throughput:

- 1) **Connection Establishment:** Three-Way Handshake;
- 2) **Slow Start:** The congestion window (CWND) regulates the number of packets in transit. The process begins with a small congestion window, doubling the number of packets sent for each received ACK until the network capacity is reached;
- 3) **Congestion Avoidance:** Once TCP achieves a certain throughput level, it prevents congestion by adjusting the transmission rate using a congestion control algorithm. During this phase, the transmission speed increases more gradually. If packet loss or high latency is detected, TCP reduces the transmission rate;
- 4) **Recovery:** If TCP detects packet loss (due to missing ACKs or duplicate ACKs), it reduces the transmission speed accordingly.

In this work, throughput was measured using *iPerf*, with the master node hosting an *iPerf* server (`iPerf -s -P 20`) and clients generating traffic via `iPerf -c <master_IP> -t 5`. Tests were conducted for 1-20 clients.

The `-P` flag in *iPerf* parameters means that the server can handle up to N connections in parallel.

With regard to the physical simulation, the test parameters that were used are the same. To launch all processes at the same time, we used the command `parallel -p 20 --noall --slf host.list 'iPerf -c Node0 -t 5'`. Parallel is used when we need to execute tasks in parallel across multiple machines. The flags regard (in order): number of parallel jobs, ensures execution of the only explicitly command, list of hostnames from the file, and the command that will be executed.

The default Linux parameters were applied using the command `iperf -c <server_ip> -t 5`, specifically: Initial Window Size 85 kB and Congestion Control Algorithm CUBIC.

IV. RESULTS

In this section, we present the main results, showing a comparison of the physical testbed and the emulated one.

For the data collection, the same test procedure was carried out for both test benches as explained in Section III-C. What we want to analyze is the ability of the switch to split the bandwidth between multiple requesting nodes, as if to simulate a massive sending of more data by the network to a centralized controller.

A. Throughput Distribution

Fig. 3 and Fig. 4 show the trend in throughput (Mbps) for different nodes (Node1-Node20) as the number of hosts changes (from 1 to 20).

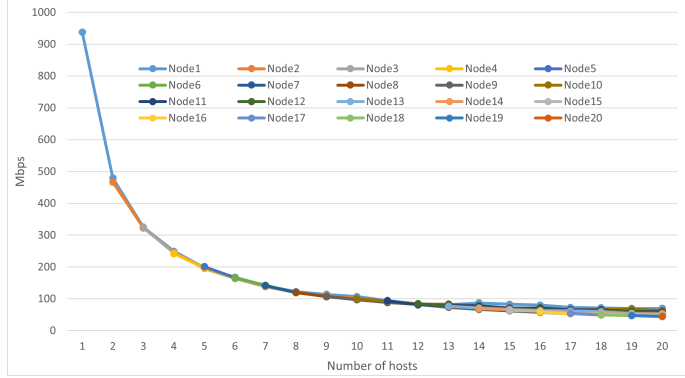


Fig. 3. Throughput of nodes in physical cluster

In both graphs, throughput decreases as the number of hosts increases, a sign that the system tends to exaggerate the maximum link capacity equally, according to the theoretical operation of the TCP protocol. However, the cluster shows more accuracy in the distribution of the capacity, while the emulated topology presents higher variability.

In the same way, Fig.5 and Fig. 6 show how the bandwidth is split with the increasing number of nodes.

Fig. 7 shows the relationship between the number of active nodes (N) and the total throughput measured in Mbps.

We can observe a gap between the performance of the physical cluster and that of the Mininet emulation.

In the physical cluster, we notice that at 20 nodes, it stabilizes around 997 Mbps, indicating that it can scale efficiently with the number of nodes, quickly reaching the maximum limit of available network capacity. In contrast, a smaller and more progressive increase in throughput is observed in Mininet, starting at about 894 Mbps and reaching 961 Mbps for 20

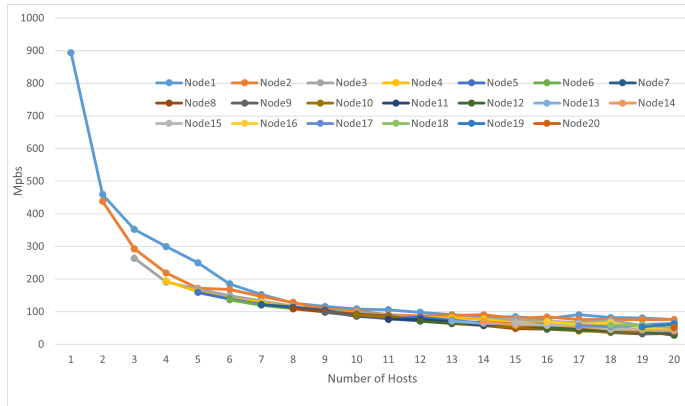


Fig. 4. Throughput of nodes in emulated cluster

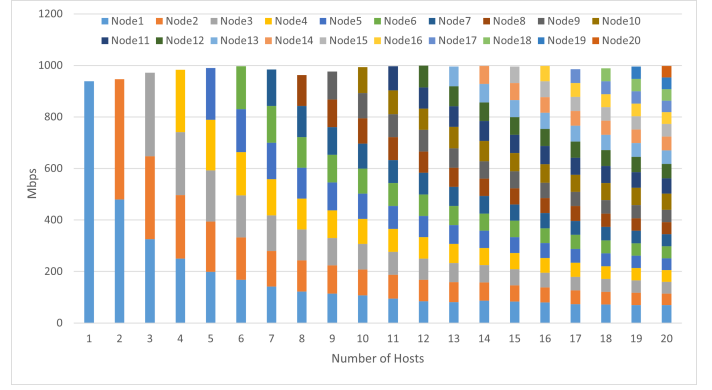


Fig. 5. Allocation of bandwidth across multiple hosts in the physical cluster

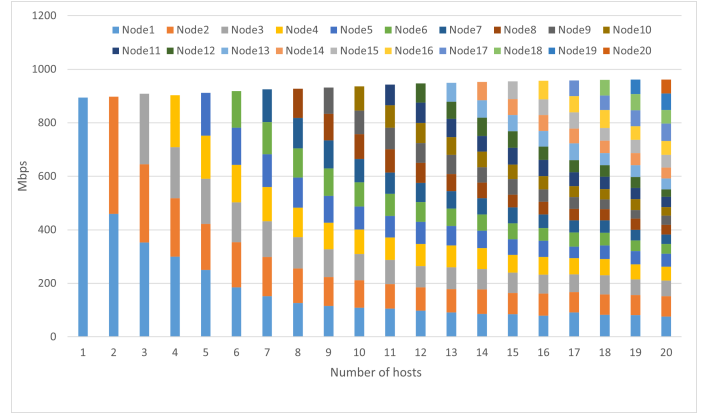


Fig. 6. Allocation of bandwidth across multiple hosts in the emulated cluster

nodes. This suggests that emulation and the presence of an SDN controller reduce overall efficiency but with a small percentage of errors, as shown in Fig. 8.

In Fig. 8, we calculated the error with respect to the total throughput of the cluster. We can notice that the error is always below 9% with a decreasing trend when a higher number of clients. Finally, Fig. 9 shows a comparison of the average per-node bandwidth in the case of $N=1,5,10,15$, and 20.

B. Mathematical Model for Throughput Degradation

Analyzing the results, we present a simplified model to predict the performance of an emulated topology with respect to a physical testbed in terms of capacity allocation.

Let R denote the theoretical maximum throughput of a physical link (e.g., 1 Gbps).

We define the total throughput $T(N)$ as

$$T(N) = R - O(N), \quad (1)$$

where $O(N)$ represents the cumulative overhead introduced by factors such as the SDN controller's CPU load and increased packet processing demands. We can assume that $T(N) \approx R$ for a physical cluster.

On the other hand, in an SDN-emulated environment such as Mininet, additional overhead factors lead to a reduction in

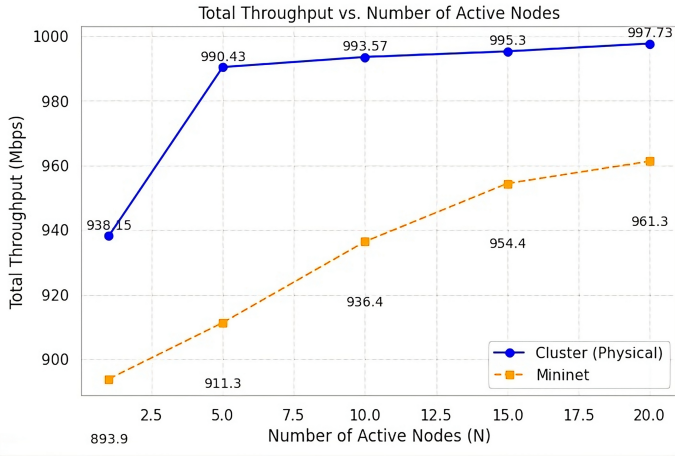


Fig. 7. Trend of Total Throughput

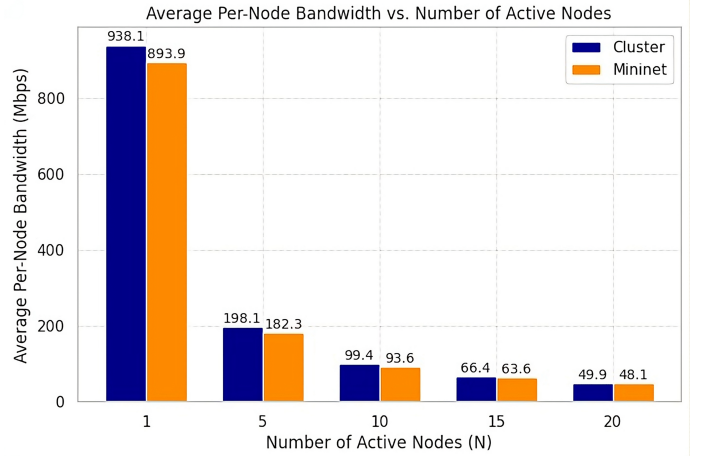


Fig. 9. Average per-node bandwidth with $N=1,5,10,15$ and 20

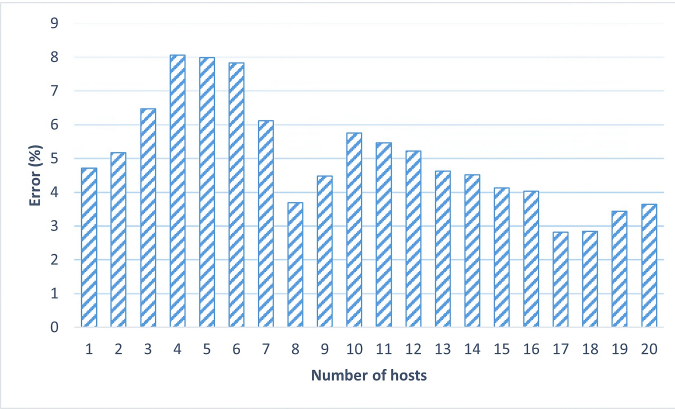


Fig. 8. Average Aggregate Throughput Percentage Error of the emulated testbed with respect to the physical one

the achievable throughput. For simplicity, we assume that this overhead scales linearly with the number of nodes:

$$O(N) = \alpha N + \beta, \quad (2)$$

with constants α and β determined empirically. Here, α captures the incremental overhead per additional node, while β represents a fixed overhead independent of N .

In a physical cluster, $O(N)$ is minimal, and thus $T(N)$ closely approximates R . In contrast, the virtualized nature of Mininet leads to higher $O(N)$ due to software-based switching and CPU contention, resulting in a noticeable degradation in $T(N)$ as N increases.

C. Consideration

Mininet's fidelity is sufficient for qualitative trends but falls short in absolute performance. Key limitations include:

- Bandwidth ceiling: Software switches cannot exceed 1 Gbps per link.
- Resource contention: Host processes share CPU resources, inflating variance.

- Asymmetric links: Mininet does not natively support split uplink/downlink bandwidths.

Our Python topology builder mitigates configuration complexity by parsing declarative files into Mininet APIs, enabling rapid reproducibility.

The findings have broader implications for the design of network digital twins and AI-driven orchestration, where deterministic emulation is essential. Future research should focus on refining the model by incorporating real-time adaptive scheduling mechanisms that mitigate performance degradation.

V. CONCLUSION

In this paper, we evaluated the fidelity and scalability of Mininet as a network emulator in comparison to a physical testbed. Our results demonstrate that Mininet provides a high degree of accuracy in emulating throughput distribution but also highlight Mininet's limitations in accurately replicating high-bandwidth network conditions due to its reliance on CPU scheduling and SDN switching. These constraints must be carefully considered when using Mininet for large-scale network simulations. Beyond the direct evaluation of Mininet, our work underscores the growing relevance of network emulation in the context of digital twin frameworks. Continuous monitoring and decision-making processes, enhancing the deployment and management of edge computing environments. Moreover, as edge computing systems demand ultra-low latency and adaptive networking strategies, leveraging network emulation within a digital twin can improve resilience and efficiency in distributed applications. Future research should explore hybrid approaches that combine Mininet with hardware-assisted emulation and AI-driven network optimizations.

ACKNOWLEDGMENT

This work has been developed within the framework of the project e. INS- Ecosystem of Innovation for Next Generation Sardinia (cod. ECS 00000038) funded by the Italian Ministry

for Research and Education (MUR) under the National Recovery and Resilience Plan (NRRP) - MISSION 4 COMPONENT 2, “From research to business” Investment 1.5, “Creation and strengthening of Ecosystems of innovation” and construction of “Territorial R&D Leaders.

APPENDIX A

A BRIEF TUTORIAL FOR REPLICATING THE TEST

We believe in the replicability of the results, and for this reason, we give a brief tutorial in order to replicate our results.

In a Linux terminal, it is possible to run the following commands:

```
git clone https://github.com/edoardoColi/
Communication_Sandbox
cd Communication_Sandbox
sudo apt install python3 python3-pip python3-
venv openvswitch-testcontroller mininet
pip3 install --upgrade pip
pip3 install mininet colorama configparser
pillow pox matplotlib ryu
sudo python3 MininetComparativeAnalysis.py
```

REFERENCES

- [1] R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, “Using mininet for emulation and prototyping software-defined networks,” *Journal of Computer Networks and Communications*, vol. 2018, pp. 1–9, 2018.
- [2] A. Tirumala, “Iperf: The tcp/udp bandwidth measurement tool,” <http://dast.nlanr.net/Projects/Iperf/>, 1999.
- [3] M. N. A. Sheikh, I.-S. Hwang, M. S. Raza, and M. S. Ab-Rahman, “A qualitative and comparative performance assessment of logically centralized sdn controllers via mininet emulator,” *Computers*, vol. 13, no. 4, p. 85, 2024.
- [4] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: Rapid prototyping for software-defined networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.
- [5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data center tcp (dctcp),” in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 63–74.
- [6] J. Lai, J. Tian, K. Zhang, Z. Yang, and D. Jiang, “Network emulation as a service (neaaS): Towards a cloud-based network emulation platform,” *Mobile Networks and Applications*, vol. 26, no. 2, pp. 766–780, 2021.
- [7] J. Ahrenholz, “Comparison of core network emulation platforms,” in *MILCOM 2010 – IEEE Military Communications Conference*, 2010, pp. 166–171.
- [8] R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, “Using mininet for emulation and prototyping software-defined networks,” in *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2014, pp. 1–6.
- [9] C. Benavices, I. García, H. Alaiz, A. Alonso, and J. M. Alija, “Networking control education by the use of software defined networking tools and techniques,” *IFAC-PapersOnLine*, vol. 48, no. 29, pp. 307–312, 2015.
- [10] S.-Y. Wang, “Comparison of sdn openflow network simulator and emulators: Estinet vs. mininet,” in *2014 IEEE Symposium on Computers and Communications (ISCC)*, 2014, pp. 1–6.
- [11] S. Hemminger, “Network emulation with netem,” in *linux.conf.au, Australia’s National Linux Conference*, Canberra, Australia, April 2005, pp. 18–23, open Source Development Lab. [Online]. Available: http://devresources.linux-foundation.org/shemminger/netem/LCA2005_paper.pdf
- [12] E. F. Kfoury, J. Gomez, J. Crichigno, and E. Bou-Harb, “An emulation-based evaluation of tcp bbrv2 alpha for wired broadband,” *Computer Communications*, vol. 161, pp. 212–224, 2020.
- [13] L. Rizzo, “Dummysnet: a simple approach to the evaluation of network protocols,” *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, 1997.
- [14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: Enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
- [15] A. Shriram, M. Murray, Y. Hyun, N. Brownlee, A. Broido, M. Fomenkov, and K. C. Claffy, “Comparison of public end-to-end bandwidth estimation tools on high-speed links,” in *Proceedings of the 6th International Workshop on Passive and Active Network Measurement (PAM 2005)*. Boston, MA, USA: Springer, 2005, pp. 306–320.