Klonet: an Easy-to-Use and Scalable Platform for Computer Networks Education

Tie Ma, Long Luo, and Hongfang Yu, *University of Electronic Science and Technology of China;* Xi Chen, *Southwest Minzu University;* Jingzhao Xie, Chongxi Ma, Yunhan Xie, Gang Sun, and Tianxi Wei, *University of Electronic Science and Technology of China;* Li Chen, *Zhongguancun Laboratory;* Yanwei Xu and Nicholas Zhang, *Theory Lab, Central Research Institute, 2012 Labs, Huawei Technologies Co., Ltd.*

https://www.usenix.org/conference/nsdi24/presentation/ma

This paper is included in the Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation.

April 16–18, 2024 • Santa Clara, CA, USA

978-1-939133-39-7

Open access to the Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation is sponsored by

Klonet: an Easy-to-Use and Scalable Platform for Computer Networks Education

Tie Ma[†], Long Luo[†], Hongfang Yu[†], Xi Chen[‡], Jingzhao Xie[†], Chongxi Ma[†],
Yunhan Xie[†], Gang Sun[†], Tianxi Wei[†], Li Chen[◇], Yanwei Xu[¶], Nicholas Zhang[¶]

[†]*University of Electronic Science and Technology of China*
[‡]*Southwest Minzu University* [◇]*Zhongguancun Laboratory*
[¶]*Theory Lab, Central Research Institute, 2012 Labs, Huawei Technologies Co., Ltd.*

Abstract

Currently, one of the simplest and most effective ways for people to gain an in-depth understanding of computer networks is through hands-on practice and experimentation on software platforms. While education is important for the field of computer networks, existing platforms are inadequate in usability and scalability, failing to fully meet all the teaching needs of computer networking education.

This paper describes our experiences in designing and using Klonet, an emulation platform for computer networking education. Klonet is *easy-to-use* for both students and tutors, which has been carefully designed to lower the barrier to use, thus making the practice more efficient. Klonet also demonstrates good *scalability*. It adopts a container-based distributed architecture and a virtual network embedding algorithm customized for this platform. Evaluation experiments show that Klonet exhibits better scalability, such as supporting more students with fewer hardware resources (*i.e.*, servers) and deploying virtual network topologies more quickly. Furthermore, to ensure stability during teaching, Klonet enhances the *robustness* of its upper orchestrator and underlying virtual networks. So far, Klonet has been adopted in 3 universities and 4 courses, serving more than 800 students. We showcase Klonet's usefulness in networking education with real use cases, including a scenario with ∼10,000 emulated routers. We also share our lessons learned from the 4 years of Klonet development and 2 years of operations.

1 Introduction

Practice and experimentation are central to the education of computer networks. Through practice, students are able to obtain hands-on experience with the complicated concepts in their computer network textbooks, thus gaining a deeper understanding of computer networks.

The educational practice of computer networks relies heavily on the field of network experiments which has been the focus of many existing works. Prior work has made great progress in three aspects: testbed [1–5], simulator [6–11], and emulator [12–37]. Testbed offers real hardware for students to manipulate, providing realistic interaction but at a higher cost. Simulator, on the other hand, uses numerical calculations to mimic hardware behavior, offering a cheaper alternative but lacking realistic interaction. Emulator is a compromise between the previous two. It uses software to simulate hardware behavior and is the most promising in education. However, as college educators ourselves, based on years of teaching experience, we found that existing emulators cannot meet our actual needs. We argue that the ideal emulator for computer network education should meet the following goals.

R1. Easy-to-use. A user-friendly educational platform should be easy to use, including getting started, mastering basic operations, *etc.*, especially for students. We know that the first step is always the hardest. If a lot of additional learning or manual configuration is required to perform relevant experiments, students may feel confused and intimidated, especially those with weak background knowledge. For tutors, an ideal platform should also be easy to use in aspects including designing, directing, correcting, and managing experiments, which can increase teaching efficiency instead of being a burden.

R2. Scalable. It also must be scalable, supporting: (1) A large number of concurrent experiments. Since students in a course usually do experiments during the same period, the platform must be able to provide services to all the students in at least one course. The more users use it, the higher the scalability requirements. Scalability requirements increase along with growing student enrollment on the platform. (2) Large-scale emulated network. To cover experiments in as many scenarios as possible, the scale of the emulated/virtual network of each experiment could be small or large. For example, a MAC address learning experiment may require less than 4 nodes, whereas routing learning in a modern datacenter network may require thousands of nodes.

To the best of our knowledge, existing network emulators are insufficient to meet the above two goals. Regarding R1, most emulators [12, 14–35, 37] require students to perform installation and complex software configurations. Considering that most undergraduate students even have not been exposed to the Linux system, this can be challenging for them and dissuade them from the computer network community. Moreover, most emulators [12, 13, 15, 16, 20–34] are initially provided for scientific research and lack consideration for graphical user interface (GUI), auxiliary experimental tools, auxiliary teaching tools, *etc.*, making students and tutors spend a lot of effort

Co-primary authors: Tie Ma and Long Luo. Hongfang Yu is the corresponding author. (Email: yuhf@uestc.edu.cn)

to use them. For R2, some emulators [12,14,16,17,20,37] can only run on a single server, which limits the scale of the virtual network it can emulate and limits the experimental scenarios it can support. Besides, some emulators [15,35] that can be deployed on multiple servers cannot properly map the virtual networks to underlying servers, causing server overloads.

To address this gap, we built Klonet, a network emulation platform to meet these educational goals. Klonet employs several technologies to make it *easy-to-use*. To minimize the difficulty of getting started, Klonet adopts a Browser/Server (B/S) architecture, allowing students and tutors to access and use it without installing any software. Klonet also provides several useful experimental tools such as traffic generator and traffic monitor for students to facilitate their experiments. In order to stimulate students' interest in learning, Klonet carefully provides a GUI that is very important for teaching. Klonet accelerates virtual network creation time, improving user experience and efficiency. For tutors, Klonet implements a container image repository that can easily add various types of nodes, making it possible to build diverse educational scenarios flexibly. Klonet includes a scene repository where students can share experiments and learn from each other, and tutors can quickly automate the creation of experiment environments. Klonet also provides experiment APIs to advanced users such as tutors to improve usability and efficiency.

To achieve *scalability*, Klonet uses light-weight containers as virtual nodes and a novel virtual network embedding algorithm to respectively reduce node overhead and link communication overhead to support more and larger virtual networks. By using container technology, Klonet increases the number of nodes that can be created on a single machine. To support large-scale virtual networks, Klonet employs a distributed orchestration architecture to create overlay virtual networks on a cluster, which enables the size of the virtual networks to scale with the cluster size. The goal of Klonet's virtual network embedding algorithm is to use as few servers as possible without overloading the servers. Klonet also implements a user management model to support multi-users.

As an open educational network emulation platform for students and tutors, *robustness* is critical. Klonet adopts methods such as periodic checks, redundant backups, and monitoring alarms to ensure the stability of its distributed orchestrator, user management model, virtual networks, and cluster.

In our evaluation, we show that Klonet can better utilize fewer physical servers to support more students' experiments faithfully. Klonet has been in development for 4 years, and we have been using it steadily in courses for 2 years. To this day, Klonet has served more than 800 students. To demonstrate Klonet's usage and benefits, we introduce the two selected use cases. The first is a course project about the Software-Defined Network (SDN) [38] and P4 [39], we use this case to showcase Klonet's usability. In the second use case, students collaborate on a routing experiment that contains up to about 10,000 nodes, demonstrating the scalability of Klonet. The lessons we have learned from developing, operating, and using Klonet in education are also presented, as a reference for building a future education platform.

In summary, our main contributions are:

- We design and implement (§4) Klonet, an easy-to-use and scalable network emulation platform for education.
- We evaluate the system performance (§5) and present use cases (§6) to demonstrate its educational applications.
- We summarize lessons (§7) learned from years of experience developing, operating, and using Klonet.

2 Related Work

Here, we discuss existing network emulation platforms used in education, categorizing them as general-purpose and education-purpose. General-purpose emulators may offer less tailored features for education, whereas education-purpose emulators prioritize educational needs. We focus on platform-level related works rather than individual components such as the Kubernetes [40] scheduler.

General-purpose network emulators. Most general emulators are not specifically designed for education but are nonetheless widely adopted in education [41].

Mininet [12] is well-known for its convenience in running small networks on laptops. However, when used in education, several challenges arise: (1) Students often struggle with the installation process. This places performance requirements on student laptops. Since the operating system of most student laptops is Windows, students need to go through the installation and use of virtual machines (VMs). This poses a significant challenge for undergraduate students who lack experience, potentially diminishing their enthusiasm for computer networks. (2) We also consider installing and configuring Mininet on servers provided by the teaching team and let students remote access. However, the original Mininet has a file isolation and scalability issue, making it unable to support multi-student. Though several follow-up variants [14–16,20–22,42] have been proposed to enhance Mininet with different capabilities such as scalability [15,42], ease of use [16], fidelity [20] and additional network scenario [22], none of the variants combining all the advantages, making it not feasible to offer an ideal network emulation service on cluster. (3) Mininet and its variants are primarily developed for research purposes and lack a student-friendly GUI, which is crucial for education.

Emulab [13] is another famous emulator that operates a dedicated cluster and provides users with network emulation service via its website, allowing users to use it without local installation. Emulab has been operated for decades and is robust. Emulab has three issues when used for education: (1) Emulab primarily focus on providing users with bare-mental servers and VMs, while has poor support for light-weight virtualization technologies like container. This makes it costly and inflexible to support courses. (2) Emulab's scalability is

Table 1: Comparison of related network emulators.

Platform Name	Easy-to-use					Scalability
	No Installation Required	GUI and Experiment API	Teaching Tools[1]	Experiment Tools[2]	Rich Node Types	
Mininet [12]	✗	Humble GUI	✗	Limited	✗	✗
Mininet-Hifi [20]	✗	Humble GUI	✗	Limited	✗	✗
Distrinet [15]	✗	No GUI	✗	✗	Limited	✓
Containernet [16]	✗	No GUI	✓	✗	Limited	✗
Vt-Mininet [21]	✗	No GUI	✗	✗	✗	✓
Mininet-Wifi [22]	✗	Humble GUI	✓	✓	✓	✗
Emulab [13]	✓	✓	Limited	✗	Limited	Limited
Netkit [37]	✗	No GUI	✓	✗	✓	✗
Kathará [17]	✗	Humble GUI	✓	✓	✓	✗
Megalos [18]	✗	Humble GUI	✓	✓	✓	✓
GNS3 [19]	✗	✓	✓	✓	✓	✓
SEED [35]	✗	✓	✓	✗	✓	✓
Mini-Internet [36]	✓	No GUI	✓	Limited	✗	✗
IPMininet [14]	✗	No GUI	✓	✓	✗	✗
Klonet (this work)	✓	✓	✓	✓	✓	✓

[1] **Teaching tools** are those designed to facilitate education, *e.g.*, Klonet's scene repository and Mini-Internet's connectivity matrix.
[2] **Experiment tools** are those designed to make experiments easier, *e.g.*, Klonet's traffic generator and IPMininet's IP configuration tools.

limited by its cluster size, and it's hard to deploy on private clusters [37]. (3) Due to the operating mode of Emulab, the physical server needs to start before creating a virtual network, resulting in a long establishment time (as shown in §5.1), which reduces the efficiency of experiments.

There are various emulators [23–27] that focus on evaluating network end-to-end properties by simplifying or abstracting network elements. However, these emulators cannot emulate all types of devices (*e.g.*, switches, routers, firewalls), limiting their ability to provide realistic interactions and serve as ideal emulators for education. Some emulators [21, 28–32] prioritize scalability at the expense of real-world performance, making it unsuitable for education. Some emulators [33, 34] are designed for specific scenarios such as enterprise networks and integrated space and terrestrial networks (ISTNs), lacking support for other network scenarios and are not easy to use for students.

Education-purpose Network emulators. Klonet is not the only network emulator proposed specifically for education. Netkit [37] exploits User-Mode Linux (UML) VM to support networking courses, the container technology chosen by Klonet is light-weight compared to the virtualization technology of Netkit. Kathará [17] is a variation of NetKit which uses Docker container instead of UML VM, but its single host network emulation architecture may cause scalability issues. Megalos [18] improves the scalability of Kathará by leveraging Kubernetes [40]. Kathará and Megalos have been used in many courses [43]. GNS3 [19] is also widely used in education and requires local software installation. Owing to reliance on VMs, GNS3 is a more resource-intensive solution compared to Klonet which employs light-weight container technology. Mini-Internet [36] is an open network emulation platform that emulates a mini version of the internet infrastructure, but it focuses on routing experiments and lacks support for other experiments. Besides, Mini-Internet does not support multi-server deployment, which may make it fail to support a large number of students. SEED emulator [35] was initially developed to support teaching network security education and can scale up the emulation across multi-server deployment. Unfortunately, it needs to embed the network manually, which may overload servers and prevent the courses from proceeding properly. IPMininet [14] enhances the IP configuration and management of the well-known emulator Mininet [12], however, it needs to be installed on the personal computers of students.

In summary, as shown in Table 1, no existing work can achieve the two educational goals of easy-to-use and scalable simultaneously except Klonet.

3 Klonet in a Nutshell

This section gives a macro view of Klonet covering its operating mode, applicable scenarios, and experiment workflow.

Operating mode. Klonet is an open platform that maintains a dedicated private or cloud cluster to offer network emulation services for courses. Its cluster is resilient, allowing operators to adjust its scale as needed. Leveraging light-weight virtualization technologies, Klonet can create (overlay) virtual networks on top of the cluster, enabling the creation of several large-scale or lots of small-scale virtual networks. Students and tutors can conveniently access Klonet via its website. Klonet stays constantly available within the campus network, serving courses during the semester and also serving self-learners. Thanks to Klonet's extensive experiment capabilities, students and tutors can efficiently conduct the majority of their experiments on a single platform.

Applicable scenarios. Klonet supports network experiments with layer 2 and above, allowing network properties

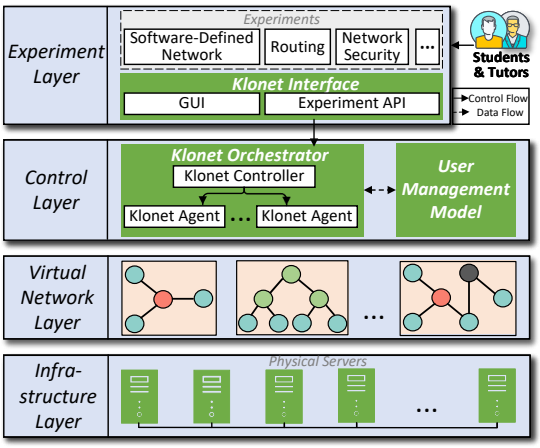


Figure 1: The overall framework of Klonet.

like latency and bandwidth on links to be configured. This versatility makes Klonet suitable for a wide range of network scenarios required in education. It can emulate data center networks, wide-area networks, large-scale enterprise networks, campus networks, *etc.* The topology of these networks can be customized, and end-to-end network experiments can also be conducted. Klonet's virtual network includes virtual Ethernet interfaces that can seamlessly connect with hardware Ethernet interfaces, supporting networks with a mixture of real and virtual devices. Klonet supports rich node types, such as (programmable) switches, routers, controllers, and hosts, enabling experiments in both data plane and control plane. Klonet supports both IPv4 and IPv6.

Experiment workflow. Conducting an experiment on Klonet involves the following steps: (1) Experiment creation. Students create experiments by dragging and dropping node icons onto the canvas, connecting nodes, and configuring basic network properties such as IP addresses. Optionally, students can choose to load pre-configured experiment templates provided by tutors. (2) Development, execution, and observation. Students develop their programs using the terminal on the Klonet web page or via Secure Shell (SSH) connections. They then run their programs and observe the resulting effects. Experiment tools such as traffic generators and monitors are available to assist students during the process. (3) Experiment completion. Once the experiment is finished, students upload their experiments to the Klonet scene repository. Tutors can use these uploaded experiments for grading purposes or allow other students to review and learn from them.

4 Design and Implementation

We first overview the framework and then describe the design adopted by Klonet to meet the educational goals.

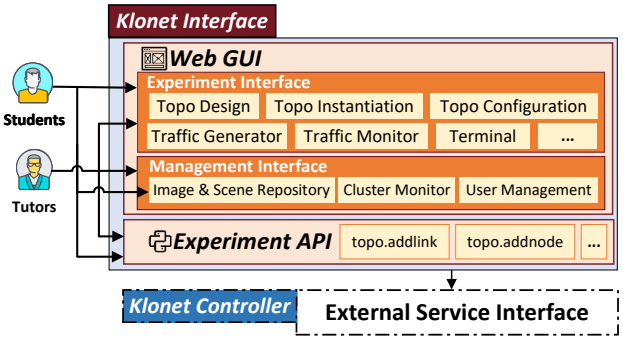


Figure 2: Klonet interface.

4.1 System Overview

As Figure 1 shows, Klonet adopts a four-layer architecture. (1) *Experiment layer:* The direct-to-user front end provides students and tutors with many easy-to-use tools via Klonet's GUI and experiment API, which include virtual network design and emulation, image repository, scene repository, traffic generator, traffic monitor, cluster monitor, *etc.* It receives user requests and communicates with the underlying layers to complete related services. (2) *Control layer:* The back end controls and manages user data, virtual networks, and clusters. This layer consists of an orchestrator distributed on each physical server and a user management model for storing user data, which together support multi-user network experiments on multi-server. (3) *Virtual network layer:* It uses light-weight virtualization technology and cross-server link technology to ensure that the virtual network can be distributed across servers. (4) *Infrastructure layer:* A cluster can provide the storage, computing, and network resources required by Klonet. It can easily remove or add new physical servers for purposes such as cost savings and launching more experiments.

4.2 Achieving Easy-to-use

Unlike research-oriented platforms, easy-to-use is particularly important for education-oriented platforms. We describe how Klonet improves ease of use with designs in terms of software architecture, user interface, reproducibility, experiment tools, and accelerating creation of virtual networks.

Browser/Server (B/S) architecture. To minimize the burden on students and reduce the difficulty in getting started, Klonet adopts a Browser/Server (B/S) architecture that allows students to use it via their browsers. To this end, Klonet is designed with two parts: front-end and back-end. The front-end is implemented in JavaScript, HTML, and CSS containing 16k lines of code. Flask [44], a web development framework is used to implement the back-end containing 45k lines of code. The front-end and back-end interact via HTTP.

Rich Interfaces. As shown in Figure 2, Klonet provides two kinds of interfaces. (1) The GUI is intuitive and is suitable for beginners such as bachelor students to perform simple operations. (2) The experiment API focuses on efficiency and

is more suitable for users with some basic network knowledge, such as master students and tutors, to conveniently and rapidly complete some repetitive and cumbersome operations.

GUI. One web page is designed for experimentation and several for management. The experiment page provides five primary features: (1) Topology design by drag-and-drop of device icons and mouse-based link connection. (2) Virtual network instantiation. (3) Configuration of network properties such as IP addresses and link bandwidths. (4) Interactive access to virtual nodes through SSH or web terminal, offering students an experience similar to real devices. Klonet helps install and configure the SSH service in the nodes. With powerful SSH-based tools (*e.g.*, VScode remote SSH extension [45]), students can efficiently complete more complex tasks. Students can also use the web terminal to conveniently perform simple tasks like running a `ping` command. Based on the Docker daemon [46], Klonet ensures real-time communication of the web terminal through websockets [47]. (5) Traffic generator and monitor tools to assist in experimental implementation. A screenshot of the experiment page is provided in Appendix A.

The management pages add much more support for experimentation. An experiment management page is designed for viewing and accessing experiments. An image/scene repository page is offered for sharing images/scenes, with tutors authorized to manage it. A user management page is designed for tutors to audit and delete student accounts conveniently. To help operators (usually tutors or TAs) quickly locate faulty servers, Klonet provides a cluster monitoring page based on Grafana [48] to view the cluster status (*e.g.*, CPU utilization).

Experiment API. This is designed to enhance the programmability. For simplicity of use, the API is implemented by encapsulating HTTP requests via Python. Listing 1 shows a simple example of using this API. Detailed API documentation is also provided to help users get started quickly.

Image repository and scene repository. Image repository and scene repository are the keys to realizing the reproducibility, including node-level and scene-level, of network experiments.

*Node-level reproducibility relies on an image repository containing node images (*i.e.*, snapshots).* Users can upload images to this repository and later instantiate them as nodes. Since the software dependencies have been packaged into the image, the applications in a node can be reproduced easily. For better usability, Klonet offers three upload methods: *commit* for saving nodes as images, *build* images via scripts, and *compressed file* for external image inclusion. Images are publicly/privately shared and indexed separately via a MySQL database [49], with entities stored in a Docker registry [50] for access across servers.

Scene-level reproducibility relies on a scene repository. We argue that a reproducible scene is a collection of code, software environment, network topology, and code execution flow. This scene repository converts all nodes of an experiment to a set of images via *commit* provided by the image repository, assigning each image a unique ID to help reestablish the corresponding node. The topology description is also saved for rebuilding virtual networks. To facilitate the scene-level reproducibility, Klonet asks users to upload a *replay script* (*i.e.*, written via experiment API) describing code execution flow. This script is optional as manual recreation is possible. MySQL is used to store image indexes, topology descriptions, and replay scripts to provide users with a reproducible scene.

```python
from klonet_api import *
# Get the available images of current student.
images = get_images()
# Select the host(ubuntu) and switch(ovs) image.
ubuntu_image = images["ubuntu"]
ovs_image = images["ovs"]
# Design our topology: h1---s1---h2.
topo = Topo()
h1 = topo.add_node(ubuntu_image, node_name="h1")
h2 = topo.add_node(ubuntu_image, node_name="h2")
s1 = topo.add_node(ovs_image, node_name="s1")
topo.add_link(h1, s1, src_IP="192.168.1.1/24")
topo.add_link(s1, h2, dst_IP="192.168.1.2/24")
# Let Klonet emulate the topology.
deploy(topo)
# Create file in h1 and h2.
exec_cmds_in_nodes(
    {"h1":["touch /log1"], "h2":["touch /log2"]})
```

Listing 1: Simple Experiment API Example.

Auxiliary useful tools. To help students focus on the network experiments themselves, Klonet provides useful tools such as traffic generator, traffic monitor, and typical topology generator. Tutors and TAs can also use these tools to complete tasks such as experimental demonstrations easily.

Traffic generator. Traffic enables the validation of diverse network scenarios. The traffic generator sources three traffic types: (1) Client-server pattern [51] traffic. (2) ON-OFF [52] characteristic traffic. (3) Customizable interval-distributed traffic, *e.g.*, Gaussian. We encapsulate an open source code from GitHub [53] to implement the first traffic type, and implement the latter two based on the Scapy library [54].

Traffic monitor. Feedback helps with experimental debugging, tuning, *etc.* Klonet develops this monitor upon libpcap [55] and reports key performance metrics, throughput, latency, and loss rate, to capture and help analyze traffic between source and destination nodes. It also provides an intuitive GUI for visualizing experimental outcomes.

Typical topology generator. Repeatedly designing some common topologies is tedious. Thus, Klonet features a generator for structures like Star, Fattree, Linear, and Tree, parameterized by variable fields. Custom topologies can also be saved as reusable templates. The typical topology generator is implemented based on the FNSS library [56].

Accelerating virtual network creation. Shortening the virtual network creation time can reduce the waiting time for users to retry experiments, improving experience and efficiency. Klonet applies two parallelization techniques: using

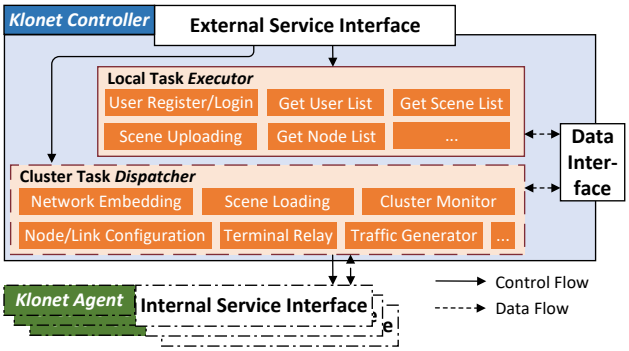


Figure 3: Klonet Controller.

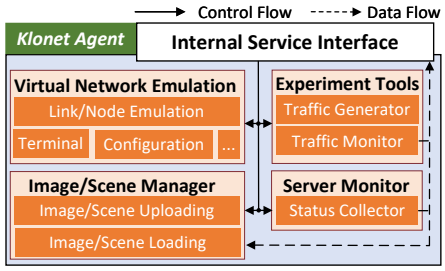


Figure 4: Klonet Agent.

a light-weight concurrency technique called Coroutines [57] to spawn virtual nodes and links concurrently within a server and permitting the orchestrator to distribute virtual network creation across physical machines for additional parallelism.

4.3 Achieving Scalability

Klonet first designs a distributed orchestrator which enables it to scale on a cluster. This poses a challenge in mapping virtual networks, which is addressed by an efficient algorithm design. For effective multi-user support, Klonet implements a user management model for data organization. To support large-scale experiments with fewer physical resources, Klonet leverages light-weight emulation technology to host as many virtual networks as possible on per physical machine.

Klonet orchestrator. Developed based on the Flask [44] framework, the orchestrator consists of a controller and several agents. The controller is installed on one server, while agents are installed on each server. Notably, the orchestrator supports single-server deployment. The controller and agents communicate with each other via HTTP. Most back-end functions of Klonet are implemented in this orchestrator.

Klonet controller. As the orchestrator's core (Figure 3), the controller receives user requests via external service interfaces and interacts with the user management model through data interfaces. It contains a local task executor for authentication and database queries, and a cluster task dispatcher for assigning *cluster tasks*, such as virtual network embedding and link configuration, to appropriate agents for execution.

Klonet agent. As shown in Figure 4, the agent receives instructions from the controller via the internal service interface. It handles on-server tasks including creating virtual networks, collecting server status, monitoring traffic, *etc.*

Virtual Network Embedding Algorithm. The Klonet controller uses a virtual network embedding (VNE) algorithm to map the virtual networks (VNs) to the substrate network (SN). For fidelity performance and efficient resource utilization, the algorithm carefully provides sufficient resources for each VN without overwhelming server capacity in the SN.

Network models and problem formulation. Each VN is modeled as an undirected graph $G = (V, E)$, with each virtual node $v \in V$ having a CPU demand $nw(v)$ and each virtual link $e \in E$ having a bandwidth demand $lw(e)$. To support cross-server deployment of large-scale VNs and efficiently use resources, Klonet partitions a VN into a set of small sub-virtual networks (sub-VNs) $subs = \{s_1, s_2, ..., s_k\}$, where $1 \le k \le |S_{server}|$. *Cut-links* represent inter-sub-VN links, and $cut(s) = \{c^s_1, c^s_2, ..., c^s_{|j|}\}, \forall c^s_i \in E, 0 \le |j| \le |E|$ denotes that connecting sub-VN s.

Let binary variable $x_{s,n}$ denote whether sub-VN s is embedded on server p_n. When embedding G to a cluster with a set S_{server} of servers, with scalability in mind, we expect to minimize the impact of the mapping of every current VN on the capabilities (potential) of the SN to deploy future large-scale VNs under resource constraints, which can be formulated as

$$\text{Minimize} \sum_{p \in S_{server}} I_p(x) \tag{1}$$

$$s.t. \quad \sum_{s \in subs} \sum_{u \in V^s} nw(u) x_{s,n} \le C(p_n), \forall p_n \in S_{server} \tag{2}$$

$$\sum_{s \in subs} \sum_{c \in cut(s)} lw(c) x_{s,n} \le B(p_n), \forall p_n \in S_{server} \tag{3}$$

$$x_{s,n} \in \{0,1\}, \forall s \in subs, p_n \in S_{server} \tag{4}$$

Eq. (1) expresses that the cluster potential is equal to the sum of all server potentials. $I(.)$ is an *Inefficiency Index (IDX)* whose value starts from 0 and grows as the use of CPU resources increases and then slowly decreases to 0 when the resources are exhausted. Besides, it increases as the use of NIC bandwidth grows. To prevent CPU overload, Ineq. (2) constrains the CPU demands of the sub-VNs deployed on server p_n to be less than its remaining CPU quota[1] $C(p_n)$. As the SN of Klonet is a local cluster where all the servers are connected by a powerful switch and each server communicates with the others through the same network interface (NIC), all substrate links connected to a server share its NIC bandwidth capacity for communication. To meet bandwidth constraints, the total bandwidth demands of the cut-links of all subnets deployed on server p_n should be less than its available NIC bandwidth $B(p)$, as expressed in (3). *This differs significantly from most existing VNE efforts, which assume exclusive bandwidth capacity per substrate link.*

[1] CPU quota is a feature of Linux Control Groups (cgroup), which controls how much CPU time a process in a container can use.

Efficient network embedding algorithm. This VNE problem is NP-Hard even with a known value of $I(\cdot)$ and the VN partition solution. A heuristic is thus employed. Through extensive experimentation, we observed that sparsely used servers often outperformed heavily used ones in hosting large VNs, given equal CPU quotas. Therefore, our algorithm is designed to minimize fragmentation by preferentially exhausting resources on previously used servers. See pseudocode in Appendix C for details.

User management model. To appropriately organize experimental information for multi-students, we build a user management model with an easily extensible architecture. We leverage the Redis database [58] to implement the management model. For advanced user and VN information, we use several tables to record them. For the low-level virtual network devices, we model them as several classes with device attributes and relationships, and instantiate devices as objects in the user management model when creating virtual networks. The detailed design can be found in Appendix B.

Virtual network emulation. Another key to scalability is the VN emulation solution. Klonet adopts Docker [59] container technology for node emulation. The reasons are (1) Container light-weight nature versus VM and better file isolation feature than the namespace technology used by emulators like Mininet [12], in line with our goal to design Klonet as a scalable and shared platform; (2) Docker container provides feature-rich node images to facilitate flexible experiment design. Default images in Klonet include: OVS [60] switch, P4 software switch [61], FRR [62] router, quagga [63] router, Ubuntu host [64], Ryu [65] controller, *etc.*

Emulated links are classified as intra-server or inter-server based on their location. Intra-server links adopt virtual Ethernet pair technology, creating virtual NICs in containers and connecting them. The virtual NICs can also be connected to real NICs for mixed virtual/realistic emulation. Inter-server links use VXLAN [66] protocol for Layer 2 connectivity across servers. Users are presented with a transparent view, unaware of distinctions between intra- and inter-server links.

To support more scenarios, Klonet uses `Linux traffic control` [67] to apply queuing disciplines on links, emulating properties, *e.g.*, bandwidth, latency, and packet loss.

4.4 Achieving Robustness

For a shared platform like Klonet, robustness plays a significant role in guaranteeing a smooth education process. We use a variety of technologies to ensure the robustness of each component of Klonet from top to bottom: Klonet orchestrator, user management model, virtual networks, and the cluster.

Robustness of the Klonet orchestrator. Klonet implements two mechanisms to ensure the availability of the orchestrator: regular requests and redundant backups.

Regular requests. Given that the Klonet controller and the Klonet agents are implemented by the web server technology, Klonet confirms that they are alive by periodically sending requests to them and getting corresponding responses. To prevent misjudgments, Klonet sends an odd number of requests at a time and uses the result of the majority of them. Once a controller or agent failure is detected, Klonet restarts it.

Table 2: Major Anomalies of Klonet.

Anomalies	Times
Cluster power outage for regular maintenance.	4
Inter-server link fails due to the firewall.	1
CPU overloaded due to broadcast storm.	1
Node boot failed due to the wrong image.	2
Node boot failed due to no startup command.	2
Lost user management model connection.	1
Image upload failed due to configuration error.	2
Inconsistent MTU among cluster servers leads to communication failures.	1

Redundant backups. During the process of Klonet restarting the controller or agent, the services they provide may temporarily fail. Therefore, Klonet designs redundant backups for them to provide services during the restart process to ensure uninterrupted services.

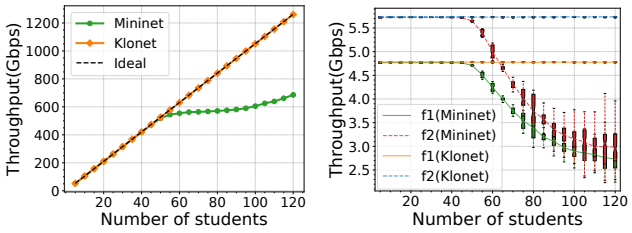
Robustness of the user management model. The methods of ensuring the robustness of the user management model are similar to those of the Klonet orchestrator. First, Klonet leverages an odd number of processes to monitor the status of the model. Second, if a failure is detected, Klonet switches to a backup. The difference is that since the user management model is a storage module, it broadcasts the received instructions to each backup to achieve data synchronization.

Robustness of the virtual networks. The virtual networks can be divided into two parts: nodes and links, and Klonet ensures their robustness respectively.

Nodes. Klonet periodically obtains the running status of each node through the docker-py [68] library. If a node is found to have stopped unexpectedly, Klonet alerts the user and tries to restart the node.

Links. For the intra-server links, they fail when one of their end nodes fails, so Klonet re-establishes the corresponding intra-server links after the nodes are restored. For the inter-server links, Klonet verifies their health through connectivity checks. As inter-server links provide layer 2 communication without requiring IP addresses for end nodes, the traditional `ping` tool does not work. Thus, we design a simple *Layer 2 ping*: Each node periodically sends data frames to the opposite end and gets a response. If the peer does not respond several times, an alert is sent to the corresponding user and operator, indicating an invalid inter-server link.

Robustness of the cluster. The large number of virtual nodes and physical servers that Klonet manages need to be monitored in a unified manner for the convenience of the operator. We use the open-source software Prometheus [69] to monitor and alert the state (CPU, memory, *etc.*) of physical servers and virtual nodes, and visualize the data centrally on the cluster monitoring page (as in §4.2) of GUI.



(a) Total throughput. (b) Throughput distribution.

Figure 5: Evaluation of multi-student concurrent experiments.

Effectiveness. Thanks to efforts in robustness, Klonet ensured the smooth running of the course for two years. During the operation, The major anomalies are shown in Table 2, and most of them were solved quickly due to Klonet's alarm and recovery mechanism.

5 System Performance

5.1 Testbed Evaluation

In this part, we first evaluate Klonet's ability to support simultaneous experiments with varying numbers of students. We also evaluate the speed of Klonet to create a virtual network, which is essential for ease of use and efficiency.

Klonet can support more students to conduct experiments simultaneously. To evaluate how many students Klonet can support to conduct experiments at the same time, we assume that at the same moment, each student is assigned a dumbbell [70] virtual network with 4 hosts and 2 switches, and runs 2 TCP flows (f1 and f2) with bottleneck link bandwidths of 5Gbps and 6Gbps respectively using `iperf3` [71] for 15 seconds. Then we vary the number of students and record the throughput of all flows. Ideally, in a high-fidelity platform, individual flow throughput should remain constant irrespective of student count, while the total throughput scales linearly with student numbers. We believe that a platform cannot support a certain number of students if the experimental results are not faithful. The evaluation is conducted on a Ubuntu 22.04 server with 2 Intel Xeon Gold 5220R CPUs and 32G RAM. To obtain comparable results, we exclude platforms that rely on heavy-weight virtualization technologies such as VM, focusing instead on comparisons with emulators that, like Klonet, use light-weight container-like virtualization techniques. With this aim, we choose the well-known emulator Mininet [12] as the benchmark.

Figure 5(a) shows the total throughput of all f1 and f2 flows, where Mininet has a significant performance drop once the number of students reaches 50, and the total throughput of all students falls short of expectations. In contrast, Klonet demonstrates nearly ideal performance. Figure 5(b) shows the throughput distribution of all flows, where each flow in Klonet maintains the expected throughput while Mininet does not. Klonet can support more students' experiments simultaneously because the node emulation technology used by Klonet has better performance isolation than that of Mininet.

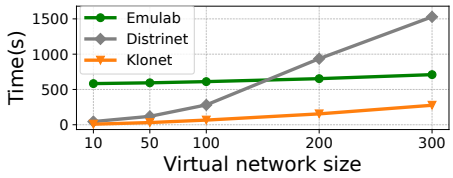
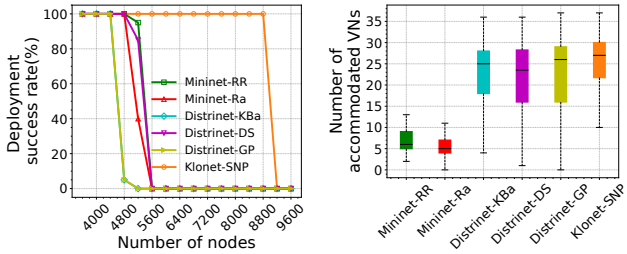


Figure 6: Virtual network creation speed.



(a) Success rate. (b) Success number.

Figure 7: Performance of Klonet's VNE algortihm(SwapNodesPartition). Compared with Mininet(RoundRobin, Random), and Distrinet(KBalance, DivideSwap, GreedyPartition)

Klonet has a faster speed to deploy VNs. We contrast Klonet's virtual networks creation time with Emulab [13] and Distrinet [15], which are representatives of the two platform operating modes: website access and local installation. To ensure consistency of the experimental environment, We request 10 *Emulab d430* bare-metal servers from Emulab and conduct the evaluation on them. We create container virtual networks of different sizes and collect the virtual networks creation time. The virtual network topology is Star, *i.e.*, multiple hosts connected to a central switch.

Figure 6 shows that the creation time for Distrinet experiences the most rapid increase as the network size escalates, mainly attributed to the utilization of LXC containers [72]. Conversely, Emulab necessitates the initiation of bare-metal servers before creating the virtual network, making it the slowest option when the network scale is below 150. In comparison, Klonet consistently exhibits the shortest deployment time up to 300 nodes. Note that the virtual node number in one student's experiment is rarely larger than 300.

5.2 Large-scale Simulation

Our VNE algorithm is one of the keys to Klonet's scalability, we use large-scale simulation to see how large and how many virtual networks it can support. We compare the proposed VNE algorithm with a range of standard algorithms employed by Mininet (cluster edition) [42] and Distrinet [73], which are two leading platforms capable of deploying large-scale virtual networks. We simulate a cluster of 10 servers, each server has

80 CPU cores with 100 CPU quota per core, and 1000Mb bandwidth capacity. To simulate network load, we randomly generate two types of topologies, FatTree and Random. The node and link resource requirements are 10 CPU quota and 10Mb, respectively. If the node or link resource requirements of a virtual network cannot be met, the its deployment is considered a failure. Note that the results of all compared VNE algorithms depend on numerical calculations, so the results are consistent in both simulation and practice.

Figure 7(a) reports the average success rate of VN deployment with varying sizes over 20 runs, decreasing as nodes increase. Mininet and Distrinet fail above 4400 nodes, while Klonet succeeds up to 8800 nodes. Figure 7(b) shows the average number of VNs accommodated over 100 runs. Klonet supports the largest number of VNs with less fluctuation in results. Distrinet ranks second, while Mininet performs worst, only one-third of Klonet. Klonet's better scalability performance benefits from its strategies of reducing cut-link weights and maximizing the potential deployment capacity of the cluster, while Distrinet only considers the former.

6 Use Cases

We have been using Klonet in our teaching practices for 2 years, and Table 3 shows the successful usages facilitated by the technical designs of Klonet.

In the following, we describe how Klonet is used in our university courses. We use two examples, one from an undergraduate-level course (Network Algorithms), and the other graduate-level (Enterprise Networks).

- **Project 1: Playing with algorithms in programmable networks:** In this project, students will learn how to implement classic shortest-path computation algorithms and compare their performance in networks with the programmable control plane or programmable data plane.
- **Project 2: Intra-domain routing:** In this graduate-level project, students will learn how to build and operate their slice of the intra-domain network. They will interconnect networks together and investigate the performance of the OSPF protocol in networks of different sizes.

6.1 Project 1: Playing with algorithms

Project Overview. Students are required to understand the basic principles of path computation algorithms in our lectures. This project helps them to gain hands-on experience with the actual performances of these algorithms in a real programmable network. We ask students to use OpenFlow [38] or P4 [39], which is widely recognized as the control plane programming protocol and data plane programming language, respectively, for implementing path computation algorithms. We request students to evaluate and compare the performance of these algorithms. Moreover, with the increasingly wide application of programmable networks in academia and industry, exposure to programmable networks itself can help students learn advanced topics in the network and engage in related research efforts. Therefore, another goal of this project is to teach basic concepts of programmable networks.

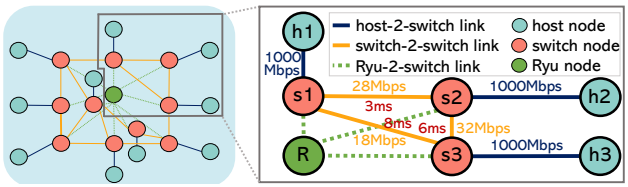


Figure 8: Topology of "Play with Alg." project.

To help students focus on algorithm-related matters rather than network configuration and operation, we use Klonet to pre-create a programmable network, then ask students just to write algorithms and feed the algorithm result into pre-written OpenFlow or P4 program templates. Specifically, since P4 is difficult for undergraduates to get started with, we only let interested students leverage the source routing [74] based on P4 to implement the result of the algorithms.

By the end, we expect the learning outcomes to be:

- Understand how OpenFlow works and master the basic operations of the OpenFlow flow table;
- Write classic path computation algorithms in OpenFlow controllers;
- Identify the effectiveness or limitations of different path computation algorithms;
- (Optional) Appreciate the difference between control plane programming and data plane programming.

To this end, we design three sub-projects in this project. Each sub-project asks students to implement one of the following classic shortest-path algorithms.

Sub-project #1: Use the Depth-First Search (DFS) algorithm to find a path with the minimum latency between two hosts.

Sub-project #2: Use the Dijkstra algorithm to find the shortest path between two hosts.

Sub-project #3: Use the Ford–Fulkerson algorithm to find the maximum network flow and the relative augmenting paths.

Figure 8 depicts the topology used, sub-project #1-2 sets links with different latencies, and sub-project #3 has different link bandwidth. Note that the switch can be an OpenFlow-enabled OpenvSwitch [60] or a P4-enabled BMv2 [61], and the controller is implemented by Ryu [65].

Using Klonet. Klonet helps students, tutors, and teaching assistants (TA) in the entire process of this 5-week project.

• Tutors use Klonet's image repository to pull new types of nodes to design new experiments quickly. Klonet gives tutors the simplicity and flexibility to update their experiments as network technology changes rapidly.

• Students use Klonet's scene repository to obtain the correct and tested version of network scenes. Compared to previous

Table 3: Klonet usages.

Grade	Use Cases	Scale	Servers[1]
2nd year undergraduate	Implement algorithm in programmable networks.	∼150stu/yr.	*S1**3
3rd year undergraduate	Networking virtual devices with Raspberry Pi.	∼40stu/yr.	*S1**1
1st year graduate	Implement network configuration such as NAT.	∼80stu/yr.	*S2**19
1st year graduate	Collaborate to achieve routing.	∼30stu/yr.	①*S1**3, *S4**14, *S5**1, ②*S3**140[2]
Self learners	Verify basic knowledge from textbooks, *etc.*	∼100stu/yr.	*S5**2

[1] Server types: *S1* (40-core 2.10GHz CPU, 256GB RAM), *S2* (2-core 2.40GHz CPU, 4GB RAM), *S3* (8-core 2.49GHz CPU, 32GB RAM), *S4* (8-core 1.70GHz CPU, 32GB RAM), *S5* (32-core 2.30GHz CPU, 128GB RAM)

[2] For this course project, we use different servers and virtual network size in each year. §6.2 describes the second-year course project.

courses without using Klonet, this dramatically reduces configuration problems for students, thus the workload of TAs.

• Klonet helps students focus on learning outcomes. TAs pre-build the VN and share it with students via the *scene repository* of Klonet before the start of each sub-project.

• Klonet allows students to interact with the elements in the networking scenario easily. It is recommended that students establish an SSH connection with the controller node to improve programming efficiency, at the same time use the *web terminal* for simple tasks, such as running the *ping* command in the host or getting the flow table of a switch. Students are also asked to write a replay script (which is just a few lines of code) and upload their sub-project into the *scene repository*.

• Klonet makes it easy for tutors and TAs to help with specific problems from students. In previous years without Klonet, tutors and TAs must deal with students' individual environments on their personal computers. Klonet standardizes the experimentation environment, thus significantly lessening the workload for teaching staff. They can directly recreate the student's networking scene and help the student solve problems.

• Klonet facilitates testing and grading of the project. After students complete a sub-project, the tutor or TA can replay the sub-project uploaded by them in the *scene repository* and grade them. Outstanding sub-projects are selected and shared in *scene repository* for students to replay and self-correct.

6.2 Project 2: Intra-domain Routing

Project overview. This graduate-level project is a hands-on experiment that involves realistic network operations. We intend to let students understand how the real network works and how network professionals operate their networks. Students are asked to build and operate an intra-domain network themselves cooperatively. To make this project realistic, we collaborated with an Internet service company to design the project. We chose the OSPF routing protocol as it is widely deployed in production networks. The students are expected to achieve the following learning outcomes:

- Build and operate their own enterprise network to have a deeper understanding of OSPF.
- Understand the importance of splitting areas.

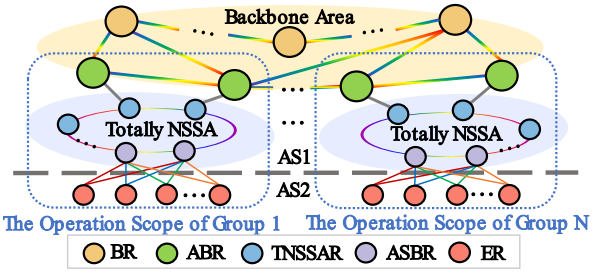


Figure 9: Topology of "Intra-domain Routing" project.

- Observe and learn the benefits of route aggregation.

Our industry collaboration partner provides the setting and configuration skeletons. Figure 9 shows the topology used in this project, with a backbone area and several Totally Not-So-Stubby-Areas (NSSA) [75]. The backbone area is the core, a mesh structure formed by routers randomly connected to each other. Some routers are called Area Border Routers (ABRs), and others are called Backbone Routers (BRs). Every two ABRs are connected to a Totally NSSA, whose topology is similar to the backbone area. Each Totally NSSA consists of Autonomous System Border Routers and Totally NSSA Routers (TNSSARs), in which every two Autonomous System Border Routers (ASBRs) connect to a number of Edge Routers (ERs) of another AS.

Using Klonet. Klonet helps students, tutors, and teaching assistants (TA) throughout this course project. We divide students into groups and let each group operate a set of routers (See Figure 9). The tutor configures BRs. Given the difficulty of this project, it is divided into three stages and each lasts for 6 weeks to configure a single area, multiple areas, and multiple areas with route aggregation, respectively.

• Klonet provides standardized and reproducible experimental settings for students and the teaching staff. The routers are implemented by the FRRouting [62] in the *image repository* of Klonet. At the outset of each stage, TAs pre-build a small VN with a total of about 500 nodes for students. Students are required to configure the routers to satisfy the stage goal and achieve network-wide connectivity. Then, students are required to modify the FRRouting configuration files by using Klonet's experiment APIs. The students are told to have scal-

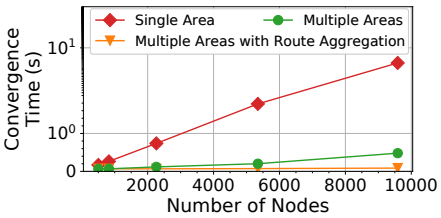


Figure 10: Convergence time of OSPF.

ability in mind, that is, when the number of routers increases or decreases, their codes should still work well.

• Klonet enables easy access to the emulated network devices. Students are mainly required to configure Router ID, Unnumbered interface, Areas and ASes, and Route aggregation to meet the goal of each stage. We have students configure the IP address of the loopback interface as the OSPF router id. The unnumbered interface can directly use the IP address of the loopback interface, thus simplifying the configuration. Students configure the NSSAs and ASes (different OSPF instances) for each router and configure route redistribution on ASBRs. Also, students configure route aggregation on the ABRs and ASBRs to aggregate routes and eliminate the communication overhead.

• Klonet makes it easy for tutors to create failure scenes in a live emulated network. At the end of each stage, the tutor creates a series of topologies of increasing size (up to ∼10,000 routers) and asks students to rerun and debug their code to build network-wide connectivity again. The tutor then takes down one link in the backbone area randomly and measures the convergence time. Finally, the tutor plots the data from the students' joint effort to show them how different network configurations affect network performance, especially convergence time.

• Klonet enables the teaching staff to develop and deploy project-specific tools. We provide a measurement tool for students to obtain the convergence time of OSPF. This tool collects logs from all routers located on each server, and by analyzing the logs, gets the start time (*i.e.*, the time that the breakdown is detected among routers) and the end time (*i.e.*, the latest time that the routing table update is completed among all routers) of the OSPF convergence process. The time synchronization scheme between servers is chrony [76]. Besides, we developed a script called `PingAll` to test the connectivity between routers. The TAs can directly update the image in the repository to distribute these tools to all students.

• The scalability of Klonet makes it possible for students to experience networks running at a realistic and modern scale. Figure 10 shows the convergence performance of different operational strategies at different network scales. Klonet is able to emulate an intra-domain network of 10,000 nodes on our cluster. The result also teaches students the importance of splitting areas and route aggregation for OSPF networks.

6.3 Feedback

To understand the reception of Klonet by students and the teaching staff, we conduct a user survey collected from more than 300 participants (67% bachelor students, 25% master students, 8% tutors and TAs). According to this survey, we have received positive feedback from most students and tutors. Most students give positive comments such as: "It helps me know the algorithms better" and "I definitely can not imagine how these algorithms work in a real network without conducting this project". The comments from teaching staff are also positive, such as: "this platform really helps quickly get started and build the experimental topology"; "it also helps save time on tedious operations and gain real experience"; "using this platform, it is easier for us to realize the experimental design of various scenarios." As mentioned in §7, there is also some negative feedback from students, and the full survey can be found in Appendix D.

The feedback confirms the need for a scalable and easy-to-use network education platform like Klonet.

7 Lessons Learnt

In this section, we introduce several lessons we have learned from developing (L1), operating (L2, L3), and using (L4, L5) Klonet in education. We believe these lessons are valuable to those seeking to establish and operate an educational network emulation platform like Klonet or use Klonet in education.

L1. Connecting to the Internet is important but has side effects. Initially, Klonet nodes had essential tools like `iperf` pre-installed, and none were connected to the Internet. However, during the teaching process, students said that they occasionally need to access the Internet for purposes like downloading Python packages. Therefore, we have all nodes connected to the Internet, which introduces a new issue: While conducting the experiment, network traffic between the two nodes did not follow the expected path specified and was routed to the bridge for the Internet connection.

To address this challenge, Klonet implemented a unique design featuring an on-off button for on-demand Internet connectivity. During the programming phase, students can toggle the button on to enable Internet connection for software installation. In contrast, they should toggle it off during experimentation to preemptively disable the Internet connection, thereby circumventing potential routing interference. It's worth noting that students can still control the node using the web terminal connected to the Docker daemon, not the Internet bridge, even when the Internet connection is offline.

L2. Resources between students need careful isolation. In our first year using Klonet for Project 1, we unexpectedly observed the *broadcast storm problem*, which exhausts CPU and memory resources, crashing the platform. This was caused by some students inadvertently flooding ARP requests to the switch with the Ryu controller. Rather than simply in-

forming students not to use ARP in ring topologies, we saw an opportunity for them to appreciate the importance of path computation algorithms. From the perspective of design, this failure highlights poor resource isolation between students. The root cause is that the container technology is not perfect in resource isolation, with all containers sharing server resources. To observe the effects of broadcast storms without crashing Klonet, we bound each container to a CPU core, limiting the impact of excessive resource usage by a single student and preventing the disruption of other experiments.

L3. Be careful when deploying Klonet or similar platforms in the public cloud. During the COVID-19 pandemic, to facilitate remote education, we deployed Klonet on the public cloud since the campus network is not accessible from the outside. However, we encountered an issue in that initially Klonet was available for a few minutes, but subsequently, broken links between its components were reported.

Upon inspection, the load balancer of the public cloud was found to terminate links between components they do not communicate within a specified period. To remedy this, we adjusted heartbeat intervals between Klonet's components to keep the link alive. Notably, unlike private clusters, these implicit mechanisms in public clouds may bring unexpected challenges.

L4. Knowledge not closely related to computer networking requires effort. Many students expressed their love for designs such as graphic design topology and web terminals, rating them as necessary, interesting and intuitive because they simplify operations and improve efficiency. However, according to student feedback, in addition to networking knowledge itself, any knowledge that is new to them (especially Linux) makes them struggle. This is overlooked in the first year. Therefore, we have supplemented and improved the introduction of relevant background knowledge in lecture notes.

L5. API access should be differentiated for students with different backgrounds. Klonet's rich APIs facilitate customizable experiments, but unrestricted access could potentially overwhelm students. For example, in our algorithm project (§6.1), all APIs are available to students initially in the first year of teaching. Due to the lack of background knowledge, students have no clear boundaries about the task they need to complete (*i.e.*, network algorithm implementation), wasting a lot of time on the preliminary work (*i.e.*, building topology and configuring link properties). We then restricted the APIs available to students in this course and let them use the built and configured topologies we provided in the scene library. With this restriction, we found that students spend significantly more time on algorithm implementation and verification, aligning better with our teaching purposes.

8 Discussion

Applications in research and even industry. The use of Klonet in research or even industry is also very encouraging. Indeed, several researchers have already adopted Klonet to carry out research works, like optimizing clock synchronization in large-scale clusters [77]. However, the research and industry field demands greater fidelity than education. For example, with our evaluation, inter-server links inevitably introduce a ms-level tail latency. While acceptable in education, such latency proves unacceptable where deterministic latency is required. Driven by the requirement from our cooperation projects with large networking companies, we are exploring techniques such as hardware offloading and latency compensation to alleviate this issue. In the future, we will make more optimizations to improve Klonet's fidelity.

Promoting Klonet's use to more courses and universities. Supported by college policy, several other courses are scheduled to use Klonet next semester. We may face more problems with large-scale use, and we will continue to maintain and optimize Klonet. Additionally, several other universities are also planning to introduce Klonet into some courses next year. However, due to many non-technical reasons, they require Klonet to be deployed on their own private cluster. This requires Klonet to be easily installed. Klonet simplifies deployment with a one-click installation script, improving ease of setup compared to Emulab [37]. In the future, we plan to offer a more convenient way for installation, using a VM image that contains Klonet and its dependencies. Technically it is easy, and popularization is more of a non-technical issue.

Limitations of applicable scenarios. While supporting a variety of scenarios, it must be acknowledged that Klonet still has limitations. For example, Klonet can only support terrestrial networks and cannot support emulating the ISTN now. In addition, Klonet uses Docker technology, which does not facilitate the modification of some Linux kernel options. For example, it cannot support experimental scenarios of adjusting the congestion control scheme in the kernel. We plan to introduce support for VMs next year to meet the different levels of configuration requirements in more scenarios.

Continuous optimizing ease of use. To ease the burden on students, we are constantly working hard to improve the ease of use of Klonet. For instance, within the laboratory, we have integrated generative AI to enhance its capabilities (*e.g.* automating topology deployment through natural language), which is currently being refined.

9 Conclusion

This paper presents Klonet, an easy-to-use and scalable network emulation platform for education. We introduce the design and implementation of Klonet. Besides, we give real use cases by using Klonet in our teaching courses which shows it has a great benefit in education. We also introduce our lessons learned from the 4 years of developing and 2 years of using Klonet in education. We plan to make Klonet available as an educational infrastructure, and we hope that Klonet can help the development of computer network education.

Acknowledgements

We would like to thank the anonymous reviewers and our shepherd Eric Eide for their constructive comments and feedback. Special thanks to Emulab for making our performance evaluation possible. We also thank Yi Gao of Linklab 2.0 for his help. We thank all members in Klonet team for their contributions, including but not limited to: Jingshan Duan, Jian Sun, Xiaoyu Yu, Chen Wang, Xunpei Hu, Yifan Su, Chang Xiao, Wei Shan, Bo Tao, Shiman Mei, Yichen He, Dongxu Wu, Jiahao Guo, Kecheng Gong, and Yifan Feng. This work is supported in part by National Key Research and Development Program of China (2021YFB3101001, 2023YFB2904600), National Natural Science Foundation of China (62102066, 62394324), Young Elite Scientists Sponsorship Program by CAST (2022QNRC001).

References

[1] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[2] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, et al. The design and operation of cloudlab. In *USENIX Annual Technical Conference*, pages 1–14, 2019.

[3] Mark Berman, Jeffrey S Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61:5–23, 2014.

[4] Wei Dong, Borui Li, Haoyu Li, Hao Wu, Kaijie Gong, Wenzhao Zhang, and Yi Gao. {LinkLab} 2.0: A multi-tenant programmable {IoT} testbed for experimentation with {Edge-Cloud} integration. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1683–1699, 2023.

[5] Jeongyoon Eo, Zhixiong Niu, Wenxue Cheng, Francis Y Yan, Rui Gao, Jorina Kardhashi, Scott Inglis, Michael Revow, Byung-Gon Chun, Peng Cheng, et al. Opennetlab: Open platform for rl-based congestion control for real-time communications. *Proc. of APNet*, 2022.

[6] The University of Washington NS-3 Consortium. Ns3 official website. https://www.nsnam.org/, 2023.

[7] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2010.

[8] Jozef Janitor, František Jakab, and Karol Kniewald. Visual learning tools for teaching/learning computer networks: Cisco networking academy and packet tracer. In *2010 Sixth international conference on networking and services*, pages 351–355. IEEE, 2010.

[9] Qingqing Yang, Xi Peng, Li Chen, Libin Liu, Jingze Zhang, Hong Xu, Baochun Li, and Gong Zhang. Deepqueuenet: towards scalable and generalized network performance estimation with packet-level visibility. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 441–457, 2022.

[10] Kaihui Gao, Li Chen, Dan Li, Vincent Liu, Xizheng Wang, Ran Zhang, and Lu Lu. Dons: Fast and affordable discrete event network simulation with automatic parallelization. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 167–181, 2023.

[11] Qizhen Zhang, Kelvin KW Ng, Charles Kazer, Shen Yan, João Sedoc, and Vincent Liu. Mimicnet: fast performance estimates for data center networks with machine learning. In *Proceedings of the ACM SIGCOMM 2021 Conference*, pages 287–304, 2021.

[12] Bob Lantz, Brandon Heller, and Nick 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*, pages 1–6, 2010.

[13] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. In *USENIX Annual Technical Conference*, 2008.

[14] Mathieu Jadin, Olivier Tilmans, Maxime Mawait, and Olivier Bonaventure. Educational virtual routing labs with ipmininet. In *ACM SIGCOMM Education Workshop*, 2020.

[15] Giuseppe Di Lena, Andrea Tomassilli, Damien Saucez, Frédéric Giroire, Thierry Turletti, and Chidung Lac. Distrinet: A mininet implementation for the cloud. *ACM SIGCOMM Computer Communication Review*, 51(1):2–9, 2021.

[16] Manuel Peuster, Johannes Kampmeyer, and Holger Karl. Containernet 2.0: A rapid prototyping platform for hybrid service function chains. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 335–337. IEEE, 2018.

[17] Gaetano Bonofiglio, Veronica Iovinella, Gabriele Lospoto, and Giuseppe Di Battista. Kathará: A container-based framework for implementing network function virtualization and software defined networks.

In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2018.

[18] Mariano Scazzariello, Lorenzo Ariemma, Giuseppe Di Battista, and Maurizio Patrignani. Megalos: A scalable architecture for the virtualization of network scenarios. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7. IEEE, 2020.

[19] Jeremy Grossman, et al. Graphical network simulator-3 (gns3). https://www.gns3.com/, 2023.

[20] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 253–264, 2012.

[21] Jiaqi Yan and Dong Jin. Vt-mininet: Virtual-time-enabled mininet for scalable and accurate software-define network emulation. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, pages 1–7, 2015.

[22] Ramon R Fontes, Samira Afzal, Samuel HB Brito, Mateus AS Santos, and Christian Esteve Rothenberg. Mininet-wifi: Emulating software-defined wireless networks. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 384–389. IEEE, 2015.

[23] Mark Carson and Darrin Santay. Nist net: a linux-based network emulation tool. *ACM SIGCOMM Computer Communication Review*, 33(3):111–126, 2003.

[24] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. *ACM SIGOPS Operating Systems Review*, 36(SI):271–284, 2002.

[25] Valerio Schiavoni, Etienne Rivière, and Pascal Felber. Splaynet: Distributed user-space topology emulation. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 62–81. Springer, 2013.

[26] Paulo Gouveia, João Neves, Carlos Segarra, Luca Liechti, Shady Issa, Valerio Schiavoni, and Miguel Matos. Kollaps: decentralized and dynamic topology emulation. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.

[27] Jiamin Cao, Ying Liu, Yu Zhou, Lin He, and Mingwei Xu. Turbonet: Faithfully emulating networks with programmable switches. *IEEE/ACM Transactions on Networking*, 2022.

[28] Dimosthenis Pediaditakis, Charalampos Rotsos, and Andrew William Moore. Faithful reproduction of network experiments. In *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 41–52, 2014.

[29] Andreas Grau, Klaus Herrmann, and Kurt Rothermel. Netplace: Efficient runtime minimization of network emulation experiments. In *Proceedings of the 2010 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'10)*, pages 265–272. IEEE, 2010.

[30] Andreas Grau, Klaus Herrmann, and Kurt Rothermel. Netbalance: Reducing the runtime of network emulation using live migration. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6. IEEE, 2011.

[31] Philip Wette, Martin Dräxler, Arne Schwabe, Felix Wallaschek, Mohammad Hassan Zahraee, and Holger Karl. Maxinet: Distributed emulation of software-defined networks. In *2014 IFIP Networking Conference*, pages 1–9. IEEE, 2014.

[32] Elias Weingärtner, Florian Schmidt, Hendrik Vom Lehn, Tobias Heer, and Klaus Wehrle. Slicetime: A platform for scalable and accurate network emulation. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, pages 253–266, 2011.

[33] Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno P Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. Crystalnet: Faithfully emulating large production networks. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 599–613, 2017.

[34] Zeqi Lai, Hewu Li, Yangtao Deng, Qian Wu, Jun Liu, Yuanjie Li, Jihao Li, Lixin Liu, Weisen Liu, and Jianping Wu. StarryNet: Empowering researchers to evaluate futuristic integrated space and terrestrial networks. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1309–1324, 2023.

[35] Wenliang Du, Honghao Zeng, and Kyungrok Won. Seed emulator: an internet emulator for research and education. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 101–107, 2022.

[36] Thomas Holterbach, Tobias Bü, Tino Rellstab, and Laurent Vanbever. An open platform to teach how the internet practically works. *ACM SIGCOMM Computer Communication Review*, 50(2):45–52, 2020.

[37] Maurizio Pizzonia and Massimo Rimondini. Netkit: network emulation for education. *Software: Practice and Experience*, 46(2):133–165, 2016.

[38] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, mar 2008.

[39] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[40] The Kubernetes Authors. Kubernetes official web site. https://kubernetes.io/, 2023.

[41] Marina Prvan and Julije Ožegović. Methods in teaching computer networks: a literature review. *ACM Transactions on Computing Education (TOCE)*, 20(3):1–35, 2020.

[42] The Mininet Project. Cluster edition prototype. https://github.com/mininet/mininet/wiki/Cluster-Edition-Prototype, 2024.

[43] Kathará. kathara success stories. https://www.kathara.org/stories.html, 2024.

[44] Flask. Flask document. https://flask.palletsprojects.com/en/2.2.x/, 2023.

[45] Microsoft. Vscode remote ssh plugin. https://code.visualstudio.com/docs/remote/ssh, 2023.

[46] Docker Inc. Docker low-level api. https://docker-py.readthedocs.io/en/stable/api.html#docker.api.exec_api.ExecApiMixin.exec_start, 2023.

[47] Ian Fette and Alexey Melnikov. Rfc 6455: The websocket protocol, 2011.

[48] Grafana Labs. Grafana official web site. https://grafana.com/, 2023.

[49] Oracle. Mysql official web site. https://www.mysql.com/, 2023.

[50] Docker Inc. Docker registry document. https://docs.docker.com/registry/, 2023.

[51] Wei Bai, Li Chen, Kai Chen, and Haitao Wu. Enabling ecn in multi-service multi-queue data centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 537–549, Santa Clara, CA, March 2016. USENIX Association.

[52] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review*, 40(1):92–99, 2010.

[53] Wei Bai, Li Chen, Kai Chen, and Haitao Wu. Trafficgenerator. https://github.com/HKUST-SING/TrafficGenerator, 2023.

[54] Scapy Community. Scapy. https://scapy.net/, 2023.

[55] The Tcpdump Group. Home | tcpdump & libpcap. https://www.tcpdump.org/, 2023.

[56] FNSS. Fnss official web site. https://fnss.github.io/, 2023.

[57] Python Software Foundation. Coroutines and tasks. https://docs.python.org/3/library/asyncio-task.html#coroutines-and-tasks, 2023.

[58] Redis Ltd. Redis official website. https://redis.io/, 2023.

[59] Docker Inc. Docker official web site. https://www.docker.com/, 2023.

[60] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of open {vSwitch}. In *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, pages 117–130, 2015.

[61] Bas Antonin and Fingerhut Andy and Sivaraman Anirudh and Arora Dushyant. Behavioral model (bmv2). https://github.com/p4lang/behavioral-model, 2023.

[62] FRRouting Project. Frrouting. https://frrouting.org/, 2023.

[63] Paul Jakma and David Lamparter. Introduction to the quagga routing suite. *IEEE Network*, 28(2):42–48, 2014.

[64] Docker Inc. Ubuntu image. https://hub.docker.com/_/ubuntu/, 2023.

[65] Ryu SDN Framework Community. Ryu controller. https://github.com/faucetsdn/ryu, 2023.

[66] Mallik Mahalingam, Dinesh Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell, and Chris Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. Technical report, 2014.

[67] Linux Foundation. tc(8) - linux manual page. https://man7.org/linux/man-pages/man8/tc.8.html, 2023.

[68] Docker Inc. Docker sdk for python. https://github.com/docker/docker-py, 2023.

[69] Prometheus Authors. Prometheus. https://prometheus.io/, 2023.

[70] P Subramanya, KS Vinayaka, HL Gururaj, and B Ramesh. Performance evaluation of high speed tcp variants in dumbbell network. *IOSR Journal of Computer Engineering*, 16(2):49–53, 2014.

[71] iPerf Group. iperf - the ultimate speed test tool for tcp, udp and sctp. https://iperf.fr/, 2023.

[72] Canonical Ltd. Linux containers. https://linuxcontainers.org, 2023.

[73] Giuseppe Di Lena, Andrea Tomassilli, Damien Saucez, Frédéric Giroire, Thierry Turletti, and Chidung Lac. Distrinet: A mininet implementation for the cloud. *ACM SIGCOMM Computer Communication Review*, 51(1):2–9, 2021.

[74] Networked Systems Group (NSG). P4-learning. https://github.com/nsg-ethz/p4-learning/tree/master/examples/source_routing, 2023.

[75] Pat Murphy. The ospf not-so-stubby area (nssa) option. Technical report, 2003.

[76] Chrony. Chrony. https://chrony.tuxfamily.org/index.html, 2023.

[77] Zhuochen Fan, Xiaodong Li, Yanwei Xu, Yuqing Li, Tong Yang, and Steve Uhlig. Work-in-progress: A novel clock synchronization system for large-scale clusters. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 519–522. IEEE, 2022.

[78] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

Appendices

A Experiment Page

Figure 11 shows the screenshot of the experiment page of Klonet's GUI. Here is a description of the various elements visible in the screenshot:

• **Klonet Dashboard.** The header at the top left indicates that this is the "Klonet Dashboard", suggesting that the platform is named "Klonet".

• **User and Experiment Information.** In the top right corner, there are details like "demo_user" and "demo_experiment", suggesting the current user's username and the name of the experiment or project they are working on, along with a "Log out" option.

• **Navigation and Action Buttons.** On the top left, there are buttons labeled "Return" and "Save", which are likely used to navigate back to a previous page and to save the current configuration, respectively.

• **Typical Topologies.** On the left sidebar, there are icons representing different network topologies such as "Tree", "Star", "Fattree", and "Linear". These could be templates for quickly setting up network structures.

• **Image Repository.** Below the Typical Topologies, there is an "Images" section with icons representing different node images. Users can upload images to this repository and later instantiate them as nodes.

• **Main Workspace.** The central area of the experiment page shows a visual representation of a network topology. Users can instantiate nodes by dragging and dropping icons from the image repository and are able to create links between nodes by connecting wires between icons. In addition, Klonet provides a control menu for each node, which can be expanded by right-clicking on the icon.

• **Control Menu of a Node.** After right-clicking the node, a contextual menu is open with options such as "Configure", "Terminal", "Delete", "File Upload", and "File Download". Users can interact with each node in the workspace to configure settings, access the web terminal, remove the element, or upload and download files.

• **Experiment Management Panel.** On the right, there is an "Experiment Management" panel where users can control the creation and destruction of the experiment, as well as "Traffic Generation" and "Traffic Measurement" sub-options to generate network traffic and monitor performance metrics such as throughput and latency of network traffic.

• **Entities Section.** Below the management panel, there's an "Entities" section with options to select "Node" or "Link", followed by a "Search" bar, which is used to filter or find specific nodes or links within the network topology. Here, users can interact with nodes (as in the "Control Menu"), delete links, and configure link properties like bandwidth.

B The Design of User Management Model

As shown in Figure 12, We assign each student a *student space* to record the experiment information. In the *student space*, there are several important components:

- The *agent_list* records the IP address of every agent.
- The *VN_list* records the name and corresponding description (*i.e.*, the information of nodes and links) of VNs of students.
- The *VN2subVN*, *subVN2agent* and *subVN_service* describe the information of the sub-VNs derived from virtual network embedding algorithm for all VNs. The *VN2subVN* records the list of sub-VN names, and users can query additional information about a sub-VN using its name. The *subVN2agent* records the IP of the server where each sub-VN is deployed. The *subVN_service* records nodes that need to start some necessary initialization programs when a VN is created.
- <*VN*>_<*node*>, <*VN*>_<*link*> and <*VN*>_<*vxlan*> are the objects of virtual devices. We use the names of VN, link, and node to index an object.

C Details of VNE Algorithm

Algorithm 1: Virtual network embedding algorithm

Input: Virtual network G, Cluster S_{server}

Output: An embedding strategy: $G \xrightarrow{mapping} S_{server}$

Compute $w_G \leftarrow$ the *sum* of the node weights of G
Compute the IDX of each server in S_{server}
if $\exists p_n \in S_{server} : C(p_n) \geq w_G$ **then**
    Embed G on the server p with the largest IDX and has a CPU capacity close to but larger than w_G;
else
    Invoke *strict_partition* (S_{server}, G)
end
Procedure `strict_partition`(S_{server}, G)
    Sort S_{server} by CPU capacity in descending order
    Select the first k' servers whose sum of CPU capacity is just greater than w_G;
    `/* k' is also the number of sub-VNs */`
    Compute the target weights $\boldsymbol{w}$ taken by the partitioning operation for sub-VNs:
    $\boldsymbol{w} = [w_1, ..., w_{k'-1}, w_{k'}] \leftarrow$
    $[\frac{C(p_1)}{w_G}, ..., \frac{C(p_{k'-1})}{w_G}, \frac{w_G - \sum_{n=1}^{k'-1} C(p_n)}{w_G}]$;
    Obtain sub-VNs: $\boldsymbol{s} \leftarrow$ Partition($G, k', \boldsymbol{w}$);
    Embed $\boldsymbol{s}$ to the selected k' servers;
    Adjust the mapping of virtual nodes to satisfy resource constraints and reduce the IDX of servers if necessary.
return *Embedding strategy of sub-VNs*

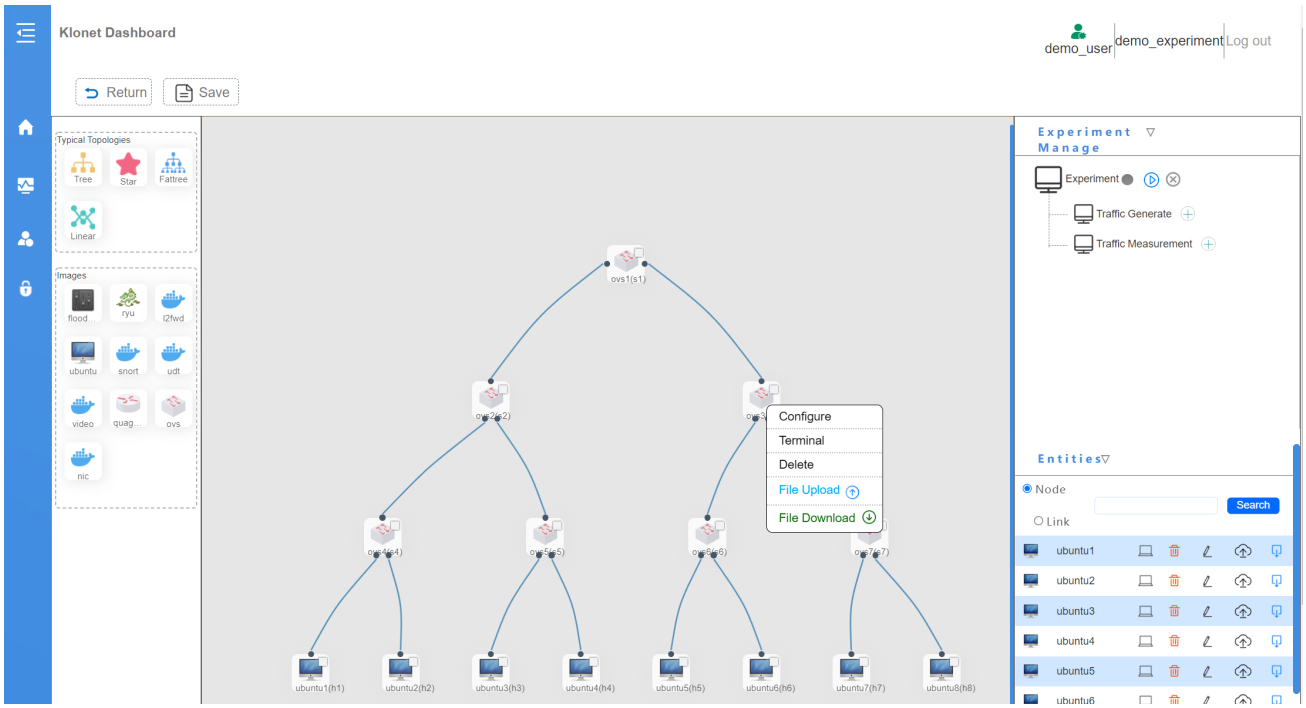


Figure 11: The Experiment Page.

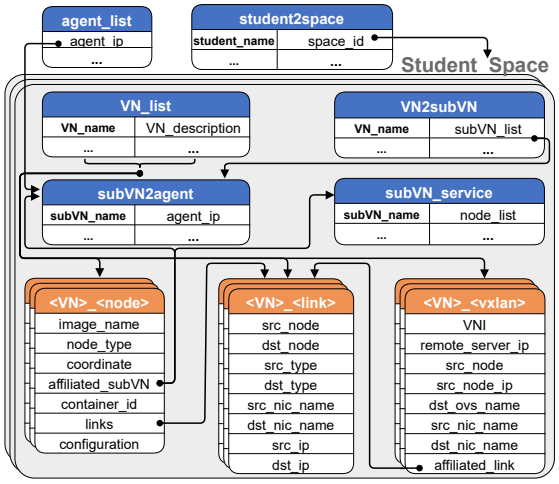


Figure 12: The User Management Model.

As shown in Algorithm 1, If a small VN can be deployed on a single server, we map it to the server with the largest IDX that satisfies the CPU capacity requirement. Otherwise, we will call the *strict_partition* function (described later) to partition this VN into a series of sub-VNs and then map these sub-VNs. To improve resource utilization, *strict_partition* first determines the number k' of sub-VNs and finds a suitable subset of candidate servers for mapping. Then, it calculates the weights $\boldsymbol{w} = [w_1, \cdots, w_{k'-1}, w_{k'}]$ of sub-VNs proportional to the capacity of candidate servers. Taking k' and $\boldsymbol{w}$ as inputs, *strict_partition* adapts a near-optimal graph partition method, METIS [78] to partition the VN into satisfied sub-VNs and then embed these sub-VNs. Finally, we adjust the mapping to meet resource constraints if it does not satisfy resource constraints and to reduce the IDX of servers if possible.

D The User Survey

We conducted an open-ended survey with the question, "What are your thoughts on the course and/or Klonet?" The survey involves 305 participants with three roles (204 bachelor students, 76 master students, 25 tutors and teaching assistants). We analyzed the questionnaire results of the three roles separately. Specifically, we conduct a coarse-grained classification of participants' answers and count the number of people with each type of answer, and then calculate the proportion of this number to the total number of people in that type of role. The results of the survey are shown in Table 4, Table 5, and Table 6.

Ethical Considerations.

- All participants in our survey gave informed and voluntary consent to the use of these data.
- Our institution's ethics review commission also consent to the conduct of this survey and its use. This work complies with all applicable ethical standards of our institution.
- We blur personal information(*e.g.* name) and group it into statistics to ensure maximizing the benefits to an individual.
- All individuals are anonymously hidden in our survey, so individual risk is minimized.
- Our survey has only one open question, so the privacy is respected.
- Also since our question is open, users cannot be induced.

Table 4: Survey result from undergraduate students.

		Comments	Number	Percentage(%)
Undergraduate Student (204)	**Positive**	Klonet is easy to get started.	43	21.08
		Can log in directly and use it very conveniently.	12	5.88
		SSH is helpful.	30	14.71
		Traffic generator or traffic monitor is helpful.	33	16.18
		Visualization is intuitive.	53	25.98
		Learn more knowledge (such as Python) after the course.	89	43.63
		Have deeper understanding of course knowledge.	157	76.96
		Acquired exemplary qualities (such as fearless in tackling difficulties,understand the importance of teamwork).	27	13.24
		Enhanced enthusiasm for learning.	30	14.71
		The course experiments are well designed.	6	2.94
		Feeling good about graphical terminals.	28	13.73
		Being able to realistically interact with network devices is fun.	14	6.86
		The TAs are very nice.	22	10.78
	Negative	Fewer documents, videos, *etc.* are available.	15	7.35
		Access is limited to the campus network; off-campus access is not available.	3	1.47
		Cannot display traffic paths in real-time.	11	5.39
		Front-end display of platform topology cannot be zoomed in and out.	2	0.98
		The first project is overly challenging as it demands a significant amount of time for comprehension of unrelated components (Python, Ryu, Xshell, ARP protocol, LLDP protocol). Despite the provision of a Ryu template, a substantial amount of time was still spent on comprehending the code within the template.	178	87.25
		The curricula and course design are too simple.	28	13.73
		There is little correlation between class content and practical content.	6	2.94

Table 5: Survey result from graduate students.

		Comments	Number	Percentage(%)
Graduate Students (76)	**Positive**	Gained a deeper understanding of OSPF or networking.	58	76.32
		Happy teamwork.	24	31.58
		Klonet is easy to get started.	10	13.16
		Acquired exemplary qualities.	9	11.84
		SSH is helpful.	21	27.63
		Enhanced enthusiasm for learning.	6	7.89
		Can log in directly and use it very conveniently.	5	6.58
	Negative	Configuring routing is difficult.	60	78.95
		Unhappy teamwork.	4	5.26
		Fewer documents, videos, *etc.* are available.	3	3.95
		Cannot use front-end for topology visualization.	11	14.47
		Experiments are too time-consuming and affect the research progress of themselves.	14	18.42
		There is little correlation between class content and practical content.	3	3.95
		A web IDE is required.	1	1.32

Table 6: Survey result from tutors and TAs.

		Comments	Number	Percentage(%)
Tutors and TAs(25)	**Positive**	Consolidated course knowledge.	20	80.00
		Fostered relationships with students.	5	20.00
		Designing experiments is easy.	19	76.00
		Like the image repository or scene repository feature.	22	88.00
		It's nice to be able to automatically scale on a cluster	3	12.00
	Negative	Experimental design is time-consuming.	6	24.00
		Takes some time to debug the software environment.	7	28.00
		Due to campus network limitations, cloud server rental may be required.	5	20.00
		Average level of student knowledge is lower than expected.	20	80.00
		Need to spend some time on offline Q&A.	8	32.00
		Lack of documentation.	2	8.00
		Configuration on the cloud is more cumbersome.	1	4.00
		Sometimes new features need to be developed based on experimental requirements.	2	8.00