SEED Emulator: An Internet Emulator for Research and Education

Wenliang Du, Honghao Zeng, Kyungrok Won
Syracuse University
Syracuse, New York, USA
{wedu,hozeng,kwon01}@syr.edu

ABSTRACT

We have developed an open-source Internet Emulator, which is a Python library, consisting of the classes for each essential element of the Internet, including autonomous system, network, host, router, BGP router, Internet exchange, etc. It also includes classes for a variety of services, including Web, DHCP, DNS, Botnet, Darknet, and Blockchain. Many other interesting network technologies can also be deployed on the emulator. Using this library, users can easily construct a miniature Internet. Although it is small, it has all the essential elements of the real Internet. The construction is compiled into Docker container files, and the emulation is executed by Docker on a single machine, or on multiple cloud machines.

This emulator has been primarily used for education since it was released in August 2021, but recently several research groups have started to use it for their research. In this paper, we present the design of this emulator and its applications. This work is still in its early stage, so the objective of this paper is to get feedback from the community, so it can be more useful to research and education.

CCS CONCEPTS

• **Networks** → **Network simulations**;

KEYWORDS

Internet emulation

ACM Reference Format:
Wenliang Du, Honghao Zeng, Kyungrok Won. 2022. SEED Emulator: An Internet Emulator for Research and Education. In *Proceedings of The 21st ACM Workshop on Hot Topics in Networks (HotNets'22).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3563766.3564097

1 INTRODUCTION

Every year since 2012, when one of the authors of this paper taught BGP and attacks in his network security class, he had to apologize to his students, because that was the only topic in his class that did not have a corresponding hands-on lab. He promised to develop one, but the next year, he had to apologize again. That apology was repeated for almost a decade. It was not that he was lazy; he simply did not know how to implement such a lab. Other than using a cyber range, the most effective way to do such a lab is to use an emulator, but an Internet emulator that could be easily used for this purpose did not exist. Although there are many existing network emulators, as we will discuss in the next section, Internet emulation and network emulation are quite different.

Eventually, after three years of development, we have built an Internet emulator called SEED emulator (its original goal was for SEcurity EDucation, hence the name SEED). While the emulator was initially developed for educational uses, a number of research groups have started to use the emulator as the platform to run their evaluation. That motivated us to expand the scope of the emulator to support research. To achieve that goal, we have been actively communicating with other researchers, listening to their needs, and developing features that support their research needs. The communication has helped shape the development of the emulator.

The SEED emulator is a Python library with 12,000 lines of code, consisting of the classes for each essential element of the Internet, including autonomous system, network, host, router, BGP router, Internet exchange, etc. It also includes the classes for a variety of services, including Web, DNS, Botnet, Darknet, Blockchain, and more are being developed. Using these classes, users can construct a mini-Internet to emulate the real-world Internet. Although it is small, it has all the essential elements of the real Internet. The construction is compiled into Docker container files, and the emulation is executed by Docker on a single machine, or on multiple cloud machines.

The SEED emulator supports many useful features. First, the emulator is extensible; plugin components can be easily added. Second, the emulator supports a hybrid emulation. External users can connect their computers to the emulator, becoming a participant of the emulation. Third, multiple emulators, local or remote, can be joined to form a larger emulation. Fourth, the entire emulation can be visualized using our visualization tool, which can also display the packet flows inside the emulator. These features are driven by the education and research needs. The objective of this paper is to present the design to a larger community, so we can hear the feedback and advice from the researchers and educators.

HotNets'22, November 14-15, 2022, Austin, Texas
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9899-2/22/11...$15.00
https://doi.org/10.1145/3563766.3564097

2 RELATED WORK AND OUR APPROACH

2.1 Comparison with Related Work

Internet emulation is not the same as network emulation. A good Internet emulator should have the following three elements: (1) network emulation, (2) Internet infrastructure emulation, and (3) service infrastructure emulation. The network emulation part covers hosts, routers, networks, and routing. This part is what most network emulators/simulators have achieved well, including mininet [10], Common Open Research Emulator (CORE) [1], NS-3 [3], GNS-3 [7], NetSim (commercial) [12], OPNET (commercial) [17], etc. Our emulator covers the basic and generic functionalities of network emulation, but compared to these existing network emulators, we have less features. For example, the control of the network bandwidth and speed is quite limited (it is feasible, but has not been implemented yet). The network emulation is necessary for our emulator, but not a focus.

The second part involves emulating the essential infrastructure of the Internet, including autonomous systems (stub and transit), Internet exchanges, BGP routing and peering. BGP (IBGP and EBGP) is the core in this part. A good emulator should allow users to easily set up and configure BGP routers and peering. BGP peering is not just about connecting two autonomous systems; a good emulator should also capture the business relationship of the peers (such as the provider-customer, and peer-to-peer relationships). Some of the existing work that covers this part include Greybox [14] and Mini-Internet project [9], but the approach that we are taking is very different and advantageous, especially on how the emulation is built and on the support of the BGP routing, peering, and configuration.

The third part is the service infrastructure emulation. One thing that makes the Internet interesting (for both research and education) is the applications (or services) running on top of it. A good Internet emulator should be able to emulate these services, especially those that involve their own infrastructure, such as DNS, Blockchain, Darknet, Botnet, Content Delivery Network (CDN), etc. Setting up these infrastructures inside an Internet emulator is non-trivial, and a good emulator should reduce the complexity for users. This is the part where our emulator really shines. 50% of the SEED emulator code is in the service part, and much more will be added in this part in the future.

2.2 Our Approach

A typical emulator consists of three parts: *composing* the emulation, *running* the emulation, and *interacting* with the emulator. The SEED emulator provides the SDK (libraries and tools) for the first and third parts, while relying on the docker container technologies to run the emulation. Figure 1 illustrates our approach.

For the emulation composition part, we have developed an open-source Python library, consisting of the classes for each essential element of the Internet, including autonomous system, network, host, router, BGP router, Internet exchange, etc. It also includes the classes for a variety of services, including Web, DNS, Botnet, Darknet, and Blockchain. Using these classes, users can easily construct their own Internet emulators using Python programs.

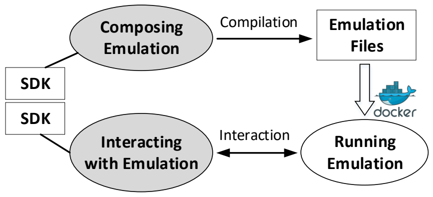

Figure 1: Our approach

The composition is eventually compiled into container files for Docker to run on a single machine or multiple cloud machines. Once the emulation has started, users can manually interact with the containers using the standard docker commands, or they can use another library that we have developed for facilitating the interaction with the emulator. Due to the page limitation, we only focus on the composition part, as it is the most significant part of the emulator.

3 EMULATING INTERNET INFRASTRUCTURE

The Internet is formed by host machines, routers, networks, Internet exchanges, and autonomous systems. The autonomous systems peer with one another using BGP. To build an Internet in an emulation, we need to provide the building blocks corresponding to these entities.

3.1 Internet Exchange

An autonomous system needs to connect to other autonomous systems, so they can exchange network traffic. Connecting two autonomous systems is called *peering*, which typically occurs inside an Internet Exchange (IX) or colocation center. An IX is basically a big high-throughput switch that connects the routers from different ASes. Through this switch, packets from one AS can be handed over to another AS. In the follow example, we create two Internet exchanges, IX-100 and IX-101 (the numbers are the autonomous system number assigned to the IXes). Internally, a network is created for each IX (their network names are automatically set to `ix100` and `ix101` in our convention.

```
ix100 = base.createInternetExchange(100)
ix101 = base.createInternetExchange(101)
```

3.2 Autonomous System

Autonomous system (AS) is an essential element of the Internet. It is emulated in our emulator. When composing the emulator, we can create an autonomous system, and then create hosts, routers, and networks inside it. In the following example, we show how to create a transit AS (with 2 as the AS number).

```
as2 = base.createAutonomousSystem(2)
as2.createNetwork("net0")
as2.createNetwork("net1")
as2.createNetwork("net2")
as2.createRouter('r0').joinNetwork('ix100')
                      .joinNetwork('net0')
as2.createRouter('r1').joinNetwork('net0')
                      .joinNetwork('ix101')
                      .joinNetwork('net1')
as2.createRouter('r2').joinNetwork('net1')
                      .joinNetwork('net2')
as2.createRouter('r3').joinNetwork('net2')
                      .joinNetwork('ix102')
```

In this example, we first create three internal networks (net0, net1, and net2). We then use four routers to connect them to form a topology (arbitrary topologies can be created). By default, these routers are configured to run the OSPF internal routing protocol. Other routing protocols can also be used. Moreover, the routers r0, r1, and r3 also connect to an Internet exchange network at IX-100, IX-101, and IX-102, respectively. Therefore, these routers are BGP routers. They will be configured to peer with one another via the Internal BGP (IBGP) protocol.

3.3 External BGP (EBGP) Peering

While the IBGP peering is automatically configured by the emulator, for EBGP peering, we need to explicitly specify which ASes peer with each other. In the following example, AS-3 peers with AS-160 and AS-161 at IX-103; their peer relationship is provider and customer, with AS-3 being the provider (i.e., AS-160 and 161 are customers of AS-3; in the real world they typically pay AS-3 for the transit service).

```
ebgp.addPrivatePeerings(103, [3],
      [160, 161], PeerRelationship.Provider)
```

In a public Internet exchange, autonomous systems may peer with many other autonomous systems. To make the peering simple, most Internet exchanges provide a special server called *route server*, so autonomous systems can peer with one another via this server. In the following example, we peer AS-2, AS-3, and AS-4 at IX-100 using a route server.

```
ebgp.addRsPeers(100, [2, 3, 4])
```

To help implement various peering relationships, we support the BGP Large communities protocol [13] in the emulator. There are many applications of the BGP Large Communities, such as identifying routes by their geographically locations (countries, continent etc.), the business relationships between peers (customers, providers, or peers), and many other aspects. In our current implementation, we only use the BGP Large communities to implement the peering relationship, but more interesting applications can be supported by our design.

The routing software used in the SEED emulator is called BIRD [4], which is open source. It supports most standard routing protocols. Configuring the BGP and internal routing is quite sophisticated and requires in-depth knowledge of how various routing protocols and how the BIRD software works. Typical users may not have such knowledge. Our emulation library hides all the complexity from the users, who can convey their needs using the provided APIs.

3.4 APIs on Host

To support various applications inside the emulator, we need to be able to install more software and files on the hosts, as well as being able to conduct configuration. While this can be done when the emulator is running, it is time-consuming to do it on many hosts. We provide APIs so these tasks can be done when we construct the emulator. The following example shows how to install software and files on nodes (containers), how to run commands when a container image is built and when the container boots up.

```
host.addSoftware('telnetd')
host.importFile(hostpath="/tmp/f.py", ...)
host.addBuildCommand('useradd -m ...')
host.appendStartCommand('/tmp/server &')
```

With these low-level APIs, users can easily customize nodes. For example, some researchers told us that they need to run a modified Ethereum program geth on some nodes. They can easily do that using these APIs. In general, anything that can be done manually on a container can be done through these APIs, which cover all the three phases of software: installation, configuration, and execution.

3.5 Hybrid Emulation

By default, the SEED Internet emulator is isolated from the outside: machines from outside cannot communicate with those inside the emulator. This closed emulation by itself has many applications, but the scope of the applications can be broaden if we allow a hybrid emulation, i.e., allowing the machines inside the Internet emulator to communicate with those on the real Internet.

With the hybrid emulation, we do not need to duplicate everything inside the emulator. For example, if there is a useful web service that we would like to include in our emulation, we do not need to duplicate that inside the emulation. Instead, we can just directly use it using our hybrid emulation technique.

To allow the emulated Internet to reach the real Internet, we need to direct the packets to an exit point, where the packets can exit from the emulator and enter the real Internet. We use BGP to achieve the goal. We create an autonomous system inside the emulator, and use this AS as the exit point. The BGP router in this AS will announce two prefixes inside the emulator: `0.0.0.0/1` and `128.0.0.0/1`. These two network prefixes cover the entire IPv4 address space, so if a destination IP address does not match with any other entry in the routing table, it will match with one of these two prefixes. This makes the exit point a default destination: if a packet does not go to any network inside the emulator, it will be routed towards the exit point. If we only want to reach certain networks in the real Internet, instead of to every network, we can narrow down the scope of the announced prefixes.

The following code snippet shows how to create and set up an exit point inside the emulator. In this example, we create an autonomous system `AS-99999`, and connect its BGP router to the Internet Exchange `ix100`.

```
as  = base.createAutonomousSystem(99999)
rt  = as.createRealWorldRouter('rw',
      prefixes=['0.0.0.0/1', '128.0.0.0/1'])
rt.joinNetwork('ix100', '10.100.0.99')
```

3.6 Participating in Emulation

External users and devices can easily participate in our emulation. In one of our experiments, we put our Internet emulator on a laptop, and hook a WiFi access point to the emulator. We brought the setup to the classroom, asking students to connect their machines or smartphones to our WiFi. By doing that, students immediately became the participants of the emulation. We call this BYOI (Bring Your Own Internet to the Class). See Figure 2.

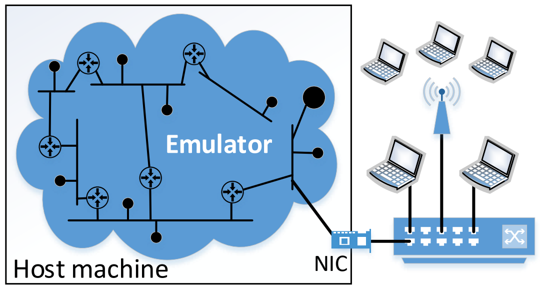

Figure 2: Participating in emulation

We then conducted the BGP network prefix hijacking inside the emulator, aiming to hijack our university's network prefix. Before the attack, students were able to access the university's website (due to our hybrid emulation setup), but as soon as the attack was launched, the university's website was no long accessible, while students could still access the rest of the Internet. Obviously, the attack only affects the routing inside the emulator, not the real world.

While this experiment was conducted for an education activity, this feature was originally requested by an IoT research group. This setting allows the researchers to easily connect all their IoT devices to the emulated Internet using WiFi or Ethernet cable, so they can observe and interfere with the behaviors of the devices. Another research group connects robots and UAVs to the emulator, so they can test their algorithms in an emulated deployment environment. We are working with this group to try to make the emulation more realistic.

3.7 Distributed Emulation

Due to the resource limitation (CPU and RAM), conducting emulation on a single machine can be limited. To solve this problem, we have developed a mechanism to easily scale up the emulation by joining multiple emulators running on different computers together. These computers can be in the same physical location or remote (on the cloud), and they each run an instance of emulator. In this section, we describe how to join these emulators together to form a larger emulation. Being able to join multiple emulators enable users to easily expand the scale of the emulation.

The merging of two emulators is done by simply bridging two Internet exchanges from the two emulators. Assume that we have two Internet emulators A and B, and we would like to join them to form a larger emulation. To achieve that, A and B should have a common Internet Exchange (say IX-100), which is emulated using a network. Therefore, A has one part of the network IX-100, and B has the other part. Bridging these two parts of the network will result in the merging of A's IX-100 and B's IX-100. See Figure 3.

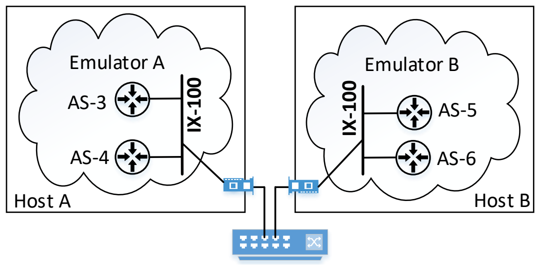

Figure 3: Bridging Internet Exchange

After the bridging, A's BGP routers on IX-100 and B's BGP routers on IX-100 are now connected to the same LAN, so they can peer with one another. Once peering is established, packets can cross the bridge from one emulator to another. Bridging can be done at multiple locations between two emulators. It can also be done by more than two emulators at the same location, as long as they share the same IX. Our emulator supports both physical bridging (using a physical switch, for machines at the same location) and virtual bridging (using Layer-2 VPN, for remote machines).

3.8 Visualization

Being able to visualize what is happening inside the Internet emulator is important. To this goal, we have developed an independent web-based tool for the visualization (called Map). This web application uses the standard docker APIs to retrieve the node and network information from the emulators, and plot them inside a web page. Users can interact with the nodes using the Map tool. When we built the emulator, we added meta data to each container, so meaningful information is preserved and can be retrieved from the emulator using the docker APIs.

From the Map, using the tcpdump filter expression, users can specify what type of packets they would like to visualize. The Map tool will then request the docker to run the tcpdump program with the filter on all the containers. An event will be reported to the Map by a container if it sees a packet matching the filter. The Map will visualize such an event. This way, we can see the packet flow. The Map also supports the record and replay feature, so users can replay events at a slower speed. A visualization example is shown in Figure 5 when we discuss an application of the SEED emulator. We are also adding new modules to this tool, so in addition to the packet-level visualization, we can also provide service-specific visualization, such as visualizing transactions on a blockchain network.

4 EMULATING INTERNET SERVICE INFRASTRUCTURE

One thing that makes the Internet interesting (for both research and education) is the applications (or services) running on top of it. Some of these services only involves standalone servers, such as Web server, DHCP server, and email server. It is quite easy to deploy this kind of services inside the emulator. However, many useful services have their own infrastructure consisting of a large number of servers, such as DNS, Blockchain, Darknet, Botnet, Content Delivery Network (CDN), etc. Setting up such an infrastructure is non-trivial, because it involves configuring many nodes and their relationships. To help users build these service infrastructures inside the SEED emulator, for each of these services, we have developed a Python class, which encapsulates the complicated setup details.

4.1 The Extensible Design

When building a service for the emulator, whether it is a standalone service or a service infrastructure, we would like this service to be independent from the underlying Internet. This way, a service is portable, and can be deployed in different Internet emulators.

We use a layered design. There are two types of layers in the SEED emulator: the base layer and the service layer. The base layer consists of hosts, networks, and routing. At this layer, each node corresponds to a "physical" entity (container). Each service layer consists of a service (or a service infrastructure). Nodes at the service layer are virtual nodes, not physical ones. Basically, a node is represented by a symbolic name. All the configuration and setup on a service node will simply use that name.

The service layer will never reference any physical node at the base layer; therefore it can be built independently from the base layer. We call each service a *component*. To deploy a service on top of a base layer, we bind the virtual nodes at the service layer to the physical nodes at the base layer, just like plugging a chip (service) onto a circuit board (base).

Currently, we have implemented several service components, including DNS, blockchain, Darknet, Botnet, Web, DHCP, and email. More components will be added in the future. In this section, we provide some details for two useful components.

4.2 Example 1: DNS Infrastructure

The goal of a DNS component (Figure 4) is to set up the DNS infrastructure, including configuring the zone files for each domain hosted in this infrastructure. Each DNS server inside this component is a virtual node. In the following code snippet, we show how we create a DNS infrastructure. The `dns.install(name)` will create a node using `name` if such a node does not exist. We can then install zones on these virtual nodes, and configure their zone files. These nodes are virtual nodes; they are just names, and they do not bind to any existing physical node in the emulator. This makes the component portable.

```
dns = DomainNameService()
dns.install('root-a').addZone('.').setMaster()
dns.install('root-b').addZone('.')
dns.install('com').addZone('com.')
dns.install('ns-example')
    .addZone('example.com.')
dns.getZone('example.com.')
    .addRecord('www A 5.5.5.5')
```

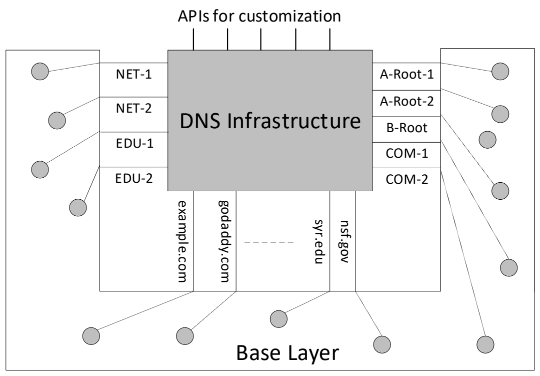

Figure 4: DNS component

The above DNS component can be saved into a file and be plugged into any base. To do that, we bind each of the virtual node in this component to a physical node in the base. After that, all the DNS configuration conducted on the virtual node will be applied to the physical node. The following example binds the virtual node `root-a` to a host in ASN-171 (the emulator will automatically find a suitable host for us, or create a new host if no suitable one is found). After the binding, this selected host will become a DNS root server.

```
emu.addBinding(Binding('root-a',
               filter=Filter(asn=171)))
```

Our DNS component supports hybrid emulation, i.e., we only need to host part of the DNS infrastructure inside the emulator, while relying on the real-world DNS for the rest. For example, we can simply host the `com` and `edu` TLDs, while relying on the real-world DNS to answer the queries for the other TLDs.

4.3 Example 2: Blockchain Emulator

Blockchain has been a very active research area these days. One of the problems faced by researchers is the evaluation. Using a real-world blockchain to do the evaluation has many issues, especially for work involving attacks. Moreover, it is very hard to know the ground truth. Many researchers have to set up their own testbed, but they face two challenges: (1) setting up a large scale test bed is quite complicated, (2) some Blockchain research [8, 15, 16] depends on the Internet infrastructure, so they need an Internet emulator.

We have developed an Ethereum-based blockchain component. In the following example, we show how to build a blockchain with 100 nodes, with one third set as bootnodes, and two third as sealers (for Proof-of-Authority consensus)

```
eth = EthereumService()
eth.setBaseConsensusMechanism
       (ConsensusMechanism.POA)
balance = 32 * pow(10, 18)
for i in range(100):
  e = eth.install("eth{}".format(i))
  if i%3 == 0:
     e.setBootNode(True)
  else:
     e.createPrefundedAccounts(balance, 1)
     e.unlockAccounts().startSealer()
```

This component has been the focus of our recent development. New APIs, driven by the requests from researchers, will be added in the future releases. On September 15th, 2022, the Ethereum Merge finally happened. We followed up with that by providing the Proof-of-Stake support in our Ethereum emulator. We plan to implement components for other blockchains, including Hyperledger [2] and Bitcoin [11].

5 EVALUATION AND APPLICATIONS

We have conducted a comprehensive evaluation on the performance of the emulator. Due to the page limitation, we are not able to include the results in this paper. Readers can find the results in a thesis written by one of the authors [18].

The Internet emulator can have many applications. Applications in cybersecurity education have already been established, while applications in research have not been established yet, but we are actively helping several research groups (in both networking and cybersecurity fields) adopt the emulator for their research. Due to the page limitation, we only describe one of the applications (in cybersecurity education). More applications can be found from the project's website [6].

In this application, we created a mini-Internet emulator using 275 containers, consisting of 5 Internet exchanges, 12 stub autonomous systems (AS), with each AS having one internal network with 20 hosts (240 hosts in total). These ASes peer with several transits ASes at the Internet exchanges. See Figure 5. During the construction of the emulator, we installed a vulnerable server on these 240 host containers. The server has a buffer-overflow vulnerability. The construction only involves 86 lines of Python code. Despite having these many machines, the entire emulator can run on a Ubuntu 20.04 virtual machine with 2 cores and 8GB of RAM, so it is possible to run on most students' personal computers.

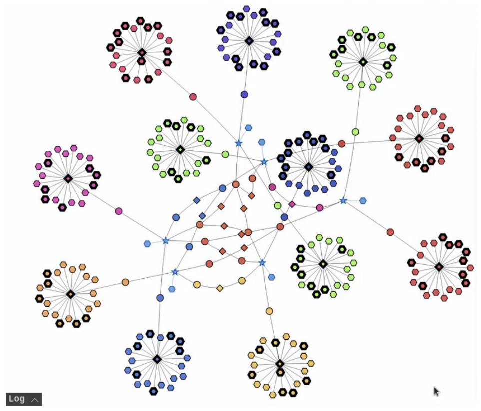

Figure 5: The spreading of the worm: the nodes with the bold black border are compromised nodes, which are also attacking others. They actually flash in visualization.

Students' job is to write a simplified Morris worm program. They can only release the worm on one of the nodes. The worm should then automatically attack the others. self duplicates, and propagates to its victim. The whole process can be visualized using our Map tool. After finishing the lab, a student commented, "seeing it from the entire Internet perspective is remarkable and stunning." A video demo can be found from YouTube [5].

6 SUMMARY

The SEED Internet emulator is an open-source project, which was developed initially for cybersecurity education, but its scope is now expanded beyond that. Its objective is to help users easily create a miniature Internet with required services deployed, so they can use the emulator to evaluate their research ideas. While the emulator is fully functioning and is already adopted by others, it is still under active development. More features will be added in the near future. We welcome other people to try the emulator, give us feedback, and help us make this open-source software to better serve the research and education communities.

The work is funded in part by the National Science Foundation (Award No. 2214916) and by the Meredith Professorship fund from the Syracuse University.

REFERENCES

[1] J. Ahrenholz, C. Danilov, T. Henderson, and J. kim. 2008. CORE: A real-time network emulator. *MILCOM 2008 - 2008 IEEE Military Communications Conference* (2008), 1–7.

[2] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolic, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. *CoRR* abs/1801.10228 (2018). arXiv:1801.10228 http://arxiv.org/abs/1801.10228

[3] NS-3 contributors. 2022. NS-3: Network Simulator. https://www.nsnam.org/. (2022).

[4] CZ.NIC. 2022. BIRD Internet Routing Daemon. https://bird.network.cz/. (2022).

[5] Wenliang Du. 2021. SEED Labs: Morris Worm Attack Lab (Demo). Available at https://www.youtube.com/watch?v=2VZV-aFoVjk. (2021).

[6] Wenliang Du. 2022. SEED Internet Emulator. Available at https://seedsecuritylabs.org/emulator/. (2022).

[7] GNS3. 2022. Graphical Network Simulator-3. https://www.gns3.com/. (2022).

[8] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., 129–144. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[9] T. Holterbach, T. Bühler, T. Rellstab, and L. Vanbever. 2020. An Open Platform to Teach How the Internet Practically Works. *SIGCOMM Comput. Commun. Rev.* (2020). https://doi.org/10.1145/3402413.3402420

[10] B. Lantz, B. Heller, and N. McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Hotnets*. Monterey, CA, USA.

[11] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. *Cryptography Mailing list at https://metzdowd.com* (03 2009).

[12] NetSim. 2022. NetSim Network Simulator. https://www.tetcos.com/. (2022).

[13] Job Snijders, John Heasley, and Martijn Schmidt. 2017. Use of BGP Large Communities. RFC 8195. (June 2017). https://doi.org/10.17487/RFC8195

[14] Gabriel L. Somlo. 2016. GreyBox: Single-Host Internet Simulator. https://github.com/cmu-sei/greybox. (2016).

[15] Muoi Tran, Inho Choi, Gi Jun Moon, Anh V. Vu, and Min Suk Kang. 2020. A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network. In *2020 IEEE Symposium on Security and Privacy (SP)*. 894–909. https://doi.org/10.1109/SP40000.2020.00027

[16] Muoi Tran, Akshaye Shenoi, , and Min Suk Kang. 2021. On the Routing-Aware Peering against Network-Eclipse Attacks in Bitcoin. In *Proceedings of the 30th USENIX Security Symposium*.

[17] Wikipedia contributors. 2021. OPNET — Wikipedia, The Free Encyclopedia. (2021). https://en.wikipedia.org/w/index.php?title=OPNET&oldid=1019617098 [Online; accessed 9-January-2022].

[18] Honghao Zeng. 2021. *SEEDEMU: The SEED Internet Emulator*. Master's thesis. Syracuse University.