

L3NS: Large Scale Network Simulation for Testing Distributed Applications

Daniil Malevanniy, Oleg Iakushkin, and Vladimir Korkhov(✉)

Saint-Petersburg State University, 7/9 Universitetskaya nab.,
St. Petersburg 199034, Russia
st048818@student.spbu.ru, {o.yakushkin,v.korkhov}@spbu.ru

Abstract. Complex distributed applications are highly depended on the quality of service of the underlying networks connecting their components. Common approaches for testing these applications may not be sufficient to ensure stability of a distributed system during networking issues. In this paper we try to explore the possibility of using lightweight virtualization for distributed application testing. Our approach is to use a set of tools provided by the Docker containerization platform to build a virtual network model of arbitrary topology with the tested application running inside. We explore scalability of this approach both when using one or more computers for modelling and try to show how the use of the developed simulation system named L3NS can allow developers to test and benchmark large-scale distributed systems e.g. blockchain networks.

Keywords: Network simulator · Containerization · Distributed applications

1 Introduction

As the rate of microprocessor performance growth slows, distributed computing technologies are growing in popularity. Distributed computation is in essence multiple computers working together to produce a single result. In most cases network connections between the computers are used to coordinate the work.

It's obvious that the development of distributed applications requires testing. Since they are highly dependent on the underlying network, its behavior must be taken into account during testing. Otherwise, successful testing can only ensure that the application will work correctly only under ideal conditions and it won't allow any assessment of the impact of network issues on the application.

The primary objective of this paper is to demonstrate the use of a virtualization for modeling dynamic network behavior and analyzing its impact on the operation of distributed applications. To ensure the operation of the virtual network, the built-in mechanisms of the OS Linux and the Docker containerization platform are used. We also evaluate the scaling potential of this solution and explore the possibility of using distributed computing to test distributed applications.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022 Workshops, LNCS 13380, pp. 639–649, 2022.
https://doi.org/10.1007/978-3-031-10542-5_44

The first part of the paper describes technologies used, general architecture and principles of the developed library. The second part describes possible scenarios for using the developed library for testing distributed applications and demonstrates its use on the example of DHT Kademlia and Ethereum blockchain using one or more machines. The third part analyzes performance of the proposed solution.

2 Related Work

There are a lot of methods for emulating the impact of network behavior on the applications using it.

Discrete event-based simulators such as NS-3 [1] and Omnet++ [2] run their own network model with an arbitrary topology and provide the user with a complete set of data about all network events occurring in the model. They are intended primarily for in-depth analysis of network resource usage and network behavior. Adapting an application for testing using discrete event modeling can be extremely labor-intensive, and the computational resources required for the model to work are unacceptable for testing large-scale applications.

There are also a number of network simulators based on the use of virtualization to simulate the operation of a computer network. Some of them, such as Eve-NG [3] and GNS-3 [4], are designed primarily for emulation and debugging of network devices and topologies and aren't suitable for application testing.

Others are aimed more to simulate the overall behavior of the network and allow application testing. Examples of such simulators include CORE [5], Mininet [6], and Kathara [7]. All of them emulate the operation of the network at the link level, provide the user with a high-level API for describing the topology for the emulated network, and allow you to run arbitrary applications in it. The main difference is the toolset used to emulate network nodes: CORE uses the Linux network namespace mechanism, Mininet uses virtual machines, and Kathara uses Docker containers. Besides that, both Mininet and CORE implement distributed simulation using multiple machines using GRE protocol.

3 Methodology

3.1 Technology Stack Description

Methods used to simulate individual computers in the network and the network interactions between them are two key aspects of the implementation of the computer network model.

We decided to use containerization technology for modeling network nodes, since it is often used as a lightweight analogue of full virtualization of computing devices. It allows you to create an independent isolated environment for individual processes using the Linux kernel toolkit, which will represent separate nodes of the network.

We used Docker [8] as a containerization platform. It provides a high-level interface for managing containers, a library of images for creating predefined environments, and a set of tools for managing containers' network environment, allowing creation of network interfaces inside that can be connected via virtual bridge link, forming virtual networks. Thus, Docker is used to manage the emulation of both network nodes and their network interactions, with the actual emulation being done via Linux OS toolset.

Another important aspect of a computer network model is the level of detail of network interactions emulation. In terms of the OSI and TCP/IP models, we can choose between emulating network events at different levels.

The purpose of the developed solution is application testing. Most applications use the application layer protocols of the TCP/IP model, utilizing transport layer resources, such as TCP and UDP ports. Because of this, we decided to model the network topology only at the transport level. This will simplify virtual networks and reduce the computational overhead. The transport layer of the TCP/IP model corresponds to the third layer of the OSI model [19], which is often denoted as L3. The focus on modeling this layer is reflected in the name of the library: "Layer 3 Network Simulator", or "L3NS" for short.

To use the resources of several machines for modeling, the Docker Swarm tool was used, which comes with Docker and allows you to centrally manage containers on different machines and create virtual networks from containers running on different machines of the cluster.

These networks are based on the use of the VXLAN protocol to transfer packets between cluster nodes and an internal routing information store that allows Docker to set up ARP tables when new containers are connected to the network. Although Docker Swarm is centrally managed, virtual networks do not use the resources of a central host after initial setup, making it easy to scale the system.

To use several machines for emulation, the Docker Swarm toolset was used. It allows one to connect several computers into a cluster with a central node, and later use the central node to manage containers on different machines and create virtual networks from containers running on different machines of the cluster.

These virtual networks use the VXLAN protocol to transfer packets between cluster nodes directly, while an internal routing information is stored on the central node and used only to set up ARP tables when new containers are connected to the network. Thus, although Docker Swarm is centrally managed, virtual networks do not use the resources of a central host after initial setup, making it easy to scale the system.

It's easy to see that when the virtual subnet combines containers running on different cluster nodes, the message delay in it will be increased relative to the subnets located within the same node. In order to limit the impact of delays on the tested application, it is necessary to minimize the number of such networks. Since a computer network structure can be viewed as a hypergraph, we can use the results of hypergraph theory. This problem is known as the problem of constructing a balanced k-way cut of a hypergraph. It is shown that this

problem is NP-complete [18]. To build a heuristic solution to this problem we use the KaHyPar algorithm [16].

To manage the quality of the virtual network, L3NS uses the tcconfig [10] utility, implemented in Python. The principle of its operation is based on the use of the tc (traffic control) [11] Linux system utility to create traffic control rules that allow you to limit bandwidth, add delay, add noise, duplicate, drop or change the order of IP packets processed by the network interface.

3.2 Implementation Details

The proposed solution for testing distributed applications is implemented as a Python library. Its main functionality is to deploy a computer network model with a distributed application running on top of it, as well as providing network state management and access to application nodes to test its functions.

The library is split into two main parts:

1. Partially abstract base classes that declare a generic interface, but only implement the functionality required to describe the structure of a virtual network, and are not tied to the choice of specific simulation technologies.
2. Classes that provide an interface implementation based on a particular virtualization technology or resources used to run the model.

At the selected transport layer of the TCP/IP model, the network topology has two main types of objects: hosts and IP subnets. These types are represented in the library by the abstract classes BaseNode and BaseSubnet, respectively. At the moment, there are two implementations for them: for running local Docker containers and networks, and for running on a Docker Swarm cluster.

In addition, there are two auxiliary base classes: Network, for managing the distribution of subnets IP addresses, which also acts as a single point of control for the operation of the model, and BaseOverlay, for the configuration of dynamic routing protocols, with implementations for RIP and OSPF provided based on the FRRouting [9] utility. The Network class does not have abstract methods and can be used directly. There is also an alternative implementation for it that uses the KaHyPar algorithm to optimize the distribution of computer network nodes among the nodes of the cluster used for modeling.

L3NS is implemented as a library in Python, so we can take advantage of the tools provided by the language ecosystem for distribution, such as the PIP package manager and the PyPI [14] package index. L3NS is published on github [12] or as a PIP package, available to be installed using one command on most Linux distributions:

```
pip3 install l3ns
```

4 Results

4.1 Evaluation Criteria

The main use case for the proposed solution is to model the operation of distributed applications in large virtual networks. Based on this, the most important

characteristics for evaluation are the efficiency of computing resources usage and the possibility of horizontal scaling when using a cluster for simulation.

To evaluate these characteristics, we will use the following metrics collected for local and distributed simulation cases for a different number of nodes in the model:

- Host CPU load
- Host RAM usage
- Container CPU load
- Container RAP usage
- Cluster leader CPU load
- Cluster leader RAM usage

For testing we used easily reproducible virtual machines running on Amazon EC2. We used one template for all of them - r5.large, which provides a dual-core processor built on the x86_64 architecture and 16 GB of RAM. We used Ubuntu 20.04 and Docker 20.10.6 across all machines.

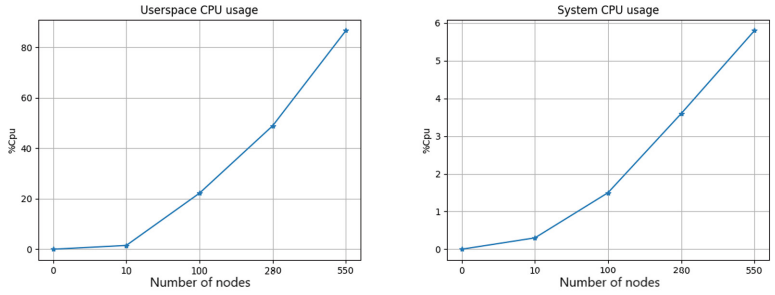
The distributed simulation cluster consists of three machines, two of which are used for simulation, and the third one is used to coordinate the work.

DHT Kademlia [13] was used as a distributed application for testing. It does not require intensive calculations to work, but at the same time it is extremely easy to scale. Thanks to this, it is possible to launch large-scale models necessary to evaluate the capabilities of the built testing system.

4.2 Local Testing

To test the capabilities of the L3NS library on a single machine, the simplest network structure is used with all nodes in a single subnet. The selected metrics were collected for networks of different sizes consequently.

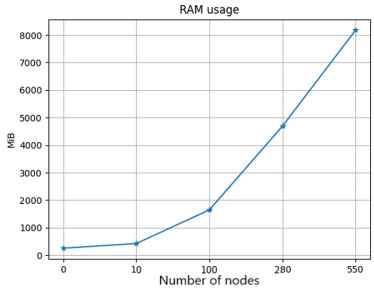
The results are presented in Fig. 1 and Fig. 2.



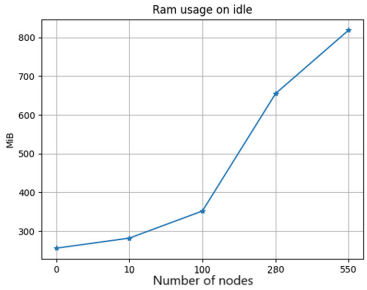
(a) Userspace CPU load (b) System CPU load

Fig. 1. CPU load during local testing.

Some preliminary findings can be drawn from those results:



(a) RAM usage while virtual network is running.



(b) RAM usage after virtual network stopped.

Fig. 2. RAM usage during local testing.

- The occupied RAM size after the simulation is shutdown grows along with the number of nodes in the model. This may indicate a possible memory leak during start or stop of containers. After restarting the Docker server, occupied RAM size returned to the initial values, indicating a possible memory leak in the Docker itself.
- The CPU load and occupied RAM size grows linearly with respect to the number of nodes. This allows us to estimate the maximum possible size of the model for local testing with a sufficiently high accuracy based on the statistics of the small-scale model.

To get a rough estimate of the computing resources consumed by network simulation, the same metrics were collected for an identical virtual network, without an application deployed inside.

The results obtained are presented in Fig. 3.

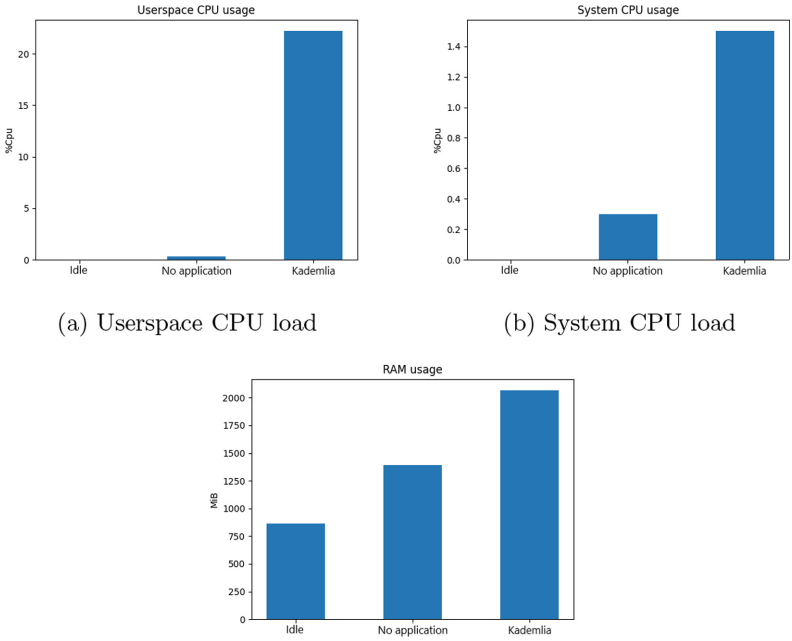
It's easy to see that even though the CPU is practically not used to maintain the network simulation, RAM usage remains high when no application is running in the model. These results are consistent with existing Docker performance analysis [17].

4.3 Cluster Testing

For testing on a cluster, a slightly more complex network topology was used, visualized in Fig. 4. To reduce the amount of traffic between two cluster nodes, the virtual network was divided into two subnets, with each one located completely on one cluster node, and two routers connecting those subnets. The routers are connected by a distributed subnet and operate using the OSPF protocol.

As before, the same metrics were gathered for models of different sizes: 100, 400 and 700 nodes. The results are presented in Fig. 5.

Following conclusions can be drawn from these results:



(a) Userspace CPU load

(b) System CPU load

(c) RAM usage

Fig. 3. Comparison of resource usage on idle, while emulating just network and network with DHT Kademlia running inside

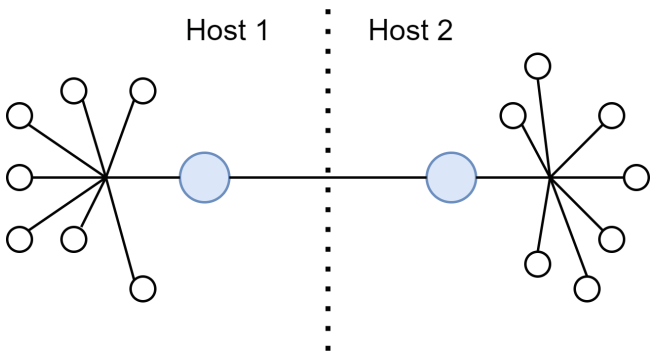
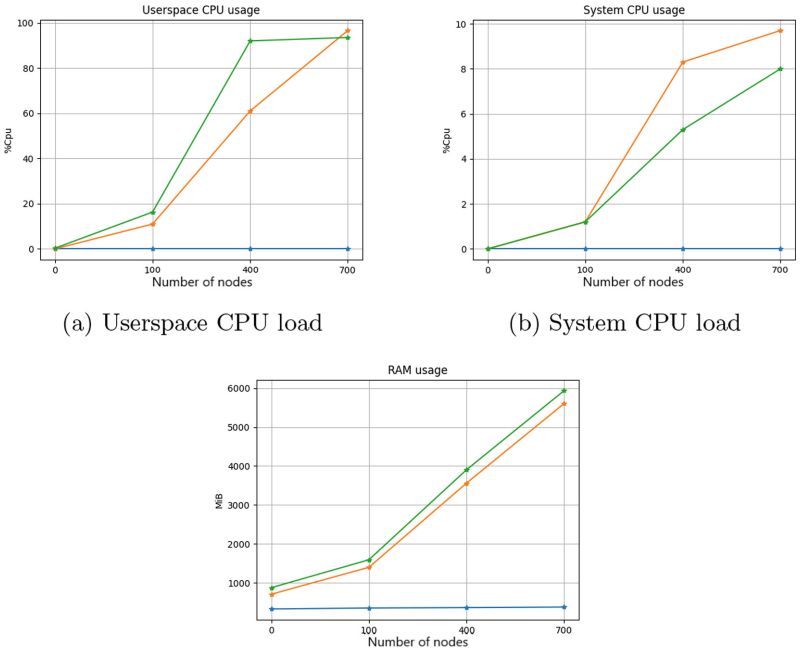


Fig. 4. Virtual network topology used in testing on cluster.

– The load on the cluster coordination node is significantly lower than the load on the modeling nodes and does not depend on the size of the model. This is due to the fact that the cluster nodes exchange traffic directly via the VXLAN protocol, and the coordination node is used only to manage the model.



(a) Userspace CPU load

(b) System CPU load

(c) RAM usage

Fig. 5. CPU and RAM usage during testing on cluster

– The CPU load of simulation nodes is higher than during local simulation even with the same number of nodes.

Despite the increased CPU load, distributed simulation allows you to run models of larger sizes than the local one.

It is worth noting that the main obstacle to further increasing the number of nodes in the model is the need to synchronously send requests to the Docker server, which takes a significant amount of time. During testing, attempting to send parallel requests permanently corrupted the state of the Docker daemon and required a restart of the Docker daemon.

4.4 Interactive Testing

To demonstrate how L3NS can be used to interactively test or demonstrate the work of distributed applications, the Ethereum [20] blockchain was chosen. When developing smart contracts or client applications for Ethereum, access to a private blockchain may simplify development and testing workflows [21].

Developers can use L3NS to deploy a private blockchain with blockchain nodes on a single computer for testing or demonstration. Since L3NS network nodes are represented by Docker containers, they work for a long time without

excessive consumption of computing power. Besides that, any container environment can be easily accessed at any time by developers.

In this example we used a network with two nodes connected by a single subnet. We used a Python script [22] to manage the virtual network, starting or stopping it depending on provided parameters.

We're using Docker built-in features to adapt virtual network model for running a blockchain: the `cpu_qouta` was used to limit the computing power available to processes running in a container. A value of 25000 corresponds to 25% of the processing power of one CPU core. Another parameter, `port`, creates a traffic forwarding rule from the host port to the container port, allowing developers external access to blockchain nodes interfaces.

Both of these options make it easier to work with the Ethereum blockchain: limiting access to the processor allows you to control the computing resources spent on mining new blocks, and external access to the container TCP port can be used to access the HTTP interface of the blockchain node to upload smart contracts to a private network or access to wallets from an external client.

4.5 Analysis

In the course of testing the functionality of the library, both its strengths and weaknesses were found. First of all, it is worth mentioning the unexpectedly high use of RAM during the operation of models and the possible memory leak that occurs at the time of starting or stopping the model.

The main positive result of testing is the demonstrated ability to scale models when running both on a cluster and on a single machine. The linear dependence of the used computing resources on the number of nodes in the model makes it possible to predict with acceptable accuracy the maximum number of nodes that can be run on one computer. In addition, it was shown that the resources of the coordinating node of the cluster are not used during the operation of the model, since all traffic exchange is decentralized. The combination of these two properties of the developed library will allow its users to calculate the cluster size required to simulate a distributed application of a given size. The main obstacle to modeling large networks is the low speed of launching models, due to the need to synchronously send to the Docker server. This problem has several possible solutions:

- Additional configuration of the Docker server for multi-threaded communication.
- Replacing containers with manual work with network namespaces.

Both solutions require a large amount of work on the library, which makes it worthwhile to conduct additional analysis of the problem to select a solution.

5 Conclusions

It's hard to overestimate the importance of testing for development or support of any application. Modern testing approaches [15] allow developers to ensure

correct behavior even in cases when application workflow involves interaction between a large number of independent components. For distributed applications, however, there is an additional layer of complexity for testing, since the interaction between its components can become unpredictable depending on the quality of service of the underlying network. We investigated the possible use of lightweight virtualization techniques for testing distributed applications in varying network conditions.

Implemented testing framework allows developers to deploy distributed applications in the virtual network of arbitrary topology and quality of service. It can be later used to benchmark the stability of the application in case of networking difficulties or for general testing and demonstrations.

Two main priorities for the framework were high scalability of virtual networks and general ease of use. While it's hard to measure the usability of the library, we discussed different aspects that make it easier for developers to use, like automation of dynamic routing configuration or distribution of network nodes across cluster nodes in case of distributed modeling.

Since L3NS mostly manages Docker and system objects, like containers and virtual bridges and doesn't participate in the work of virtual network, it can play the role of an interactive environment manager, as it was demonstrated in the last example. Using L3NS in this way can simplify the process of developing and manually testing distributed applications.

In this paper we were mostly focused on analyzing the scalability of our approach. Results show that both in case of local and distributed modeling the size of the virtual network can be increased linearly along with the amount of the computational power available. In our preliminary testing we managed to launch virtual networks with up to 10 000 nodes in them.

Main limiting factor for scaling of networks is the amount of computational resources required for the tested application. If it can be roughly approximated, it's possible to predict required resources, acquire them with any cloud service provider and launch a virtual network of an arbitrary size.

References

1. Riley, G.F., Henderson, T.R.: The ns-3 network simulator. In: Modeling and Tools for Network Simulation, pp. 15–34. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12331-3_2
2. Varga, A.: OMNeT++. In: Modeling and Tools for Network Simulation, pp. 35–59. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12331-3_3
3. EVE-NG: https://www.eve-ng.net/
4. Al-Ani, D.R., Al-Ani, A.R.: The performance of IPv4 and IPv6 in terms of routing protocols using GNS 3 simulator. Procedia Comput. Sci. **130**, 1051–1056 (2018)
5. Ahrenholz, J.: Comparison of CORE network emulation platforms. In: 2010-Milcom 2010 Military Communications Conference. IEEE (2010)
6. De Oliveira, R.L.S., et al.: Using Mininet for emulation and prototyping software-defined networks. In: 2014 IEEE Colombian Conference on Communications and Computing (COLCOM). IEEE (2014)

7. Bonofiglio, G., et al.: 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. IEEE (2018)
8. Docker: Enterprise Application Container Platform. https://docker.org
9. FRRouting Project. https://frrouting.org/
10. Hombashi, T.: tcconfig: A tc command wrapper. https://tcconfig.readthedocs.io/en/latest/
11. Stanic, M.P.: Tc-traffic control. Linux QOS Control Tool (2001)
12. L3NS GitHub repository. https://github.com/rukmarr/l3ns
13. Maymounkov, P., Mazières, D.: Kademlia: a peer-to-peer information system based on the XOR metric. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45748-8_5
14. L3 Network Simulator. https://pypi.org/project/l3ns/
15. Leung, H.K., White, L.: A study of integration testing and software regression at the integration level. In: Proceedings. Conference on Software Maintenance 1990, pp. 290–301. IEEE, November 1990
16. Schlag, S.: High-quality hypergraph partitioning. Doctoral dissertation, Karlsruhe Institute of Technology, Germany (2020)
17. Felter, W., et al.: An updated performance comparison of virtual machines and Linux containers. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE (2015)
18. Lyaudet, L.: NP-hard and linear variants of hypergraph partitioning. Theor. Comput. Sci. **411**(1), 10–21 (2010)
19. Alani, M.M.: Guide to OSI and TCP/IP models (2014)
20. Dannen, C.: Introducing Ethereum and solidity, vol. 1, pp. 159–160. Apress, Berkeley (2017)
21. Hu, B., et al.: A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems. Patterns **2**(2), 100179 (2021)
22. L3NS GitHub repository, Etherum examples. https://github.com/rukmarr/l3ns/blob/master/examples/eth/netmanager.py