Modelling the Interaction of Distributed Service Systems Components

Oleg Iakushkin(✉), Daniil Malevanniy, Ekaterina Pavlova, and Anna Fatkina

Saint-Petersburg University, 7/9 Universitetskaya nab., St. Petersburg 199034, Russia
o.yakushkin@spbu.ru

Abstract. The development of efficient distributed service systems places particular demands on interaction testing methods. It is required to consider not only the work of individual components at each stage of development but also their interaction as a system. Often, such testing is performed directly at the time of publishing the application with end-users. In this paper, we consider methods of emulating and testing distributed systems on limited resources available to individual developers. What to do with factors beyond the scope of tests of one application? How would the system behave when the network is unstable, and the third-party service does not work correctly? In this work, we present a new tool created to simulate the operation of a full-fledged service system and failure of its specific elements. The main feature of the solution we created is in the provision of abstraction for the developer, which does not require a deep understanding of the work of network infrastructures. It allows us to compare the work of several distributed applications under the same conditions, check the stability of the programme regarding the unstable operation of network elements, study the influence of the network structure on the of services behaviour and detect their vulnerabilities associated with interservice interaction.

Keywords: Network · Modeling systems · Docker · Distributed applications

1 Introduction

Large systems require rigorous testing before commissioning. Let us consider the distributed application, in addition to the correct operation of each of its components, and it is necessary to take into account network interactions. In this work, we focus on testing of such communications.

There are several modelling tools [8] (for example, GNS3 [2], eve-ng [1]) with which you can deploy a network of virtual machines and test the application on it. However, these testing systems require the user to know in the field of system administration.

O. Iakushkin—This research was partially supported by the Russian Foundation for Basic Research grant (project no. 17-29-04288).

Applications can be tested on real users using tools like Pumba [6] or Gremlin [3], but such a solution carries the risk of losing users' interest in the service.

The main goal of our work is to provide an easy way to model a scalable network. An application developer does not require in-depth knowledge of network technology to use the tools described in this paper. We propose to consider a model of a network consisting of Docker image nodes and routers. This approach allows emulation of a distributed system locally without going into the details of network interactions. The model enables reproducing effects observed by developers, such as duplication, loss, delay, or distortion of data packages.

2 Related Work

There are about two dozen emulators of distributed networks [24]. Most of them are free and available in the public domain (including GitHub). Also, HP, Huawei, Cisco have their network emulators oriented to work with equipment of a specific manufacturer. We considered the most popular network modelling environments. We selected some characteristics for comparison (Table 1): ease of use, scalability, and versatility. The ease of use of the tool is evaluated by the following criteria: configuration format, the availability of visualization, automatic deployment, and launch of distributed applications, the ability to access applications on the Internet.

There is no universal solution for emulating large-scale systems. In some cases, preference is given to modelling process details of the network layer L2; in other cases, the interaction is considered at the level of model abstractions. L3NS [4] and MADT [5] solutions allow while preserving the complexity of the services under consideration, to simplify the network abstraction as much as possible, providing the availability of modelling on one or several computing nodes and platforms.

3 Problem Definition

Our goal is to create a universal, scalable solution with which you can quickly simulate the interaction of network components and monitor or change the state of the model in real-time.

To emulate a virtual computer network, the Docker containerization [18] platform is used. Moreover, when the model is run on a cluster WireGuard [15] is used to model IP subnets. Also, a set of FRRouting utilities is used, which allows the implementation of dynamic routing protocols, and the Tcconfig shell, which allows to control [14] the quality of the network dynamically and which is used to improve the realism of the model. The hypergraph separation algorithm KaHyPar is used to organize distributed modelling of virtual [20] computer networks. The jgraph, Flask, and zmq messaging libraries are used to operate the user interface (Fig. 1).

Table 1. Comparative table of existing modeling technologies

Options/ Competitors	L3NS	Madt	Netkit/ Kathara	Mininet	Marionnet	zimulator (VNUML)	Eve-NG	NS3	GNS3	Omnet++ / Omnest
Network configuration format	Python Script	Python Script/ JSON	BASH script / special config	Python API	project file with conf of vm	VNUML Script	labs	c++ script	virual network editor	NED topology description language
Use of lightweight virtualization [7]	+	+	+	- VirtualBox, Qemu, VMware	- VirtualBox, VMware	-	-	+	-	-
Visualization	-	+	+ Netkit Lab Generator	+	+	-	+	+	+	+
Automatically deploy and launch distributed applications	+	+	The application must be correctly installed in the host system	+	+	+	+	+	-	-
The ability to scale the system by using more than 1 physical node for deployment	+	-	-	+	-	-	+ for a fee	+	+	+
Network simulation level	L3	L3	L3	L2	L2	DES*	L2	L2	L2	DES
Connection of external network nodes	+	-	-	+	+	-	+	-	+	-
The ability to access applications on the Internet	+	+	+	+	+	-	+	+	+	+
Platforms on which simulated applications can run	Linux	Linux	all platforms	Unix/ Linux	GNU/ Linux	Linux	VM based Windows, Linux, network devices	Linux, MacOS, FreeBSD	VM based Windows, Linux, host device	all platforms where a modern C++ compiler is available
Open source	MIT	Academic Public License	GPL3	MIT derevative	GPL	GPL3	Academic Public License	GPL2	GPL	Academic Public License

*DES – Discrete-event simulation, not bound to networking level.

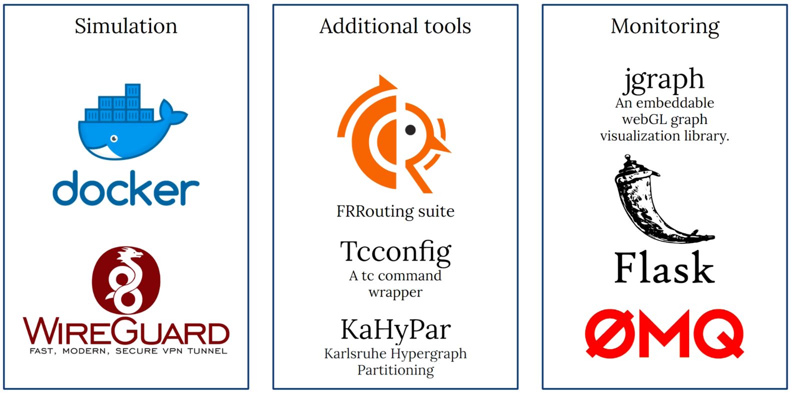


Fig. 1. Technology stack used by MADT and L3NS.

4 Proposed Architecture

The traditional way of modelling computing nodes of a network [12] is the use of hypervisors that manage virtual machines. The kernels of the operating systems are separated; this provides isolation for applications [22]. In case of modelling a network of interacting application services, nodes and applications do not need isolation at the kernel level of the OS, isolation of the network stack that the computing node [11] use is sufficient.

The Linux kernel provides a tool (Namespaces) [13] that allows isolation of the low-level environments [17]. A container is an isolated set of namespaces on the top of which an application is running. The Docker configuration platform allows to create such environments corresponds to predefined templates and run applications in them. Besides, using Docker, we can combine network interfaces

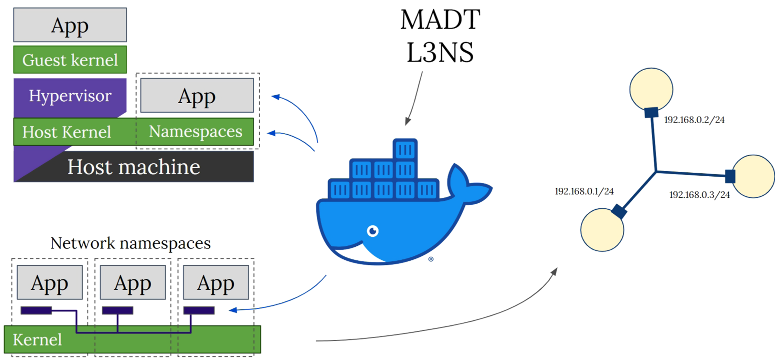


Fig. 2. MADT/L3NS workflow: 1. Commands are sent directly to docker server; 2. Docker server creates containers using Linux namespaces; 3. Docker connects containers with virtual LANs; 4. Virtual LANs and docker containers form a virtual global network.

located in different network namespaces with a virtual bridge into one common IP subnet, the hosts of which are containers. Our solutions allow pre-setting of the network topology and launch parameters of containers. This network is launched by Docker [25] and it turns out a virtual network model that fully works on one computer (Fig. 2). If the simulated application requires a large number of resources (Fig. 3), it is advisable to divide the simulation into several computers. This solves the problem of efficiently sharing the load across hosts to minimize latency in virtual networks connecting containers [16] located on different hosts. In each of the containers launched as part of the model, a Unix socket is mounted. This allows sending messages to the monitoring system [26], even if the network interfaces of the container are blocked or have a long delay, thus ensuring stable monitoring [9]. Using the web interface, you can monitor the state of the model in real-time and interactively manage network parameters for each node, controlling [19] delays, losses, duplicate packets.

5 Modeling Algorithm

The work proceeds according to the following algorithm:

1. Using the API in Python, a "Laboratory" is created — a description of the network model, which includes a list of network nodes, their distribution over subnets, and startup parameters.

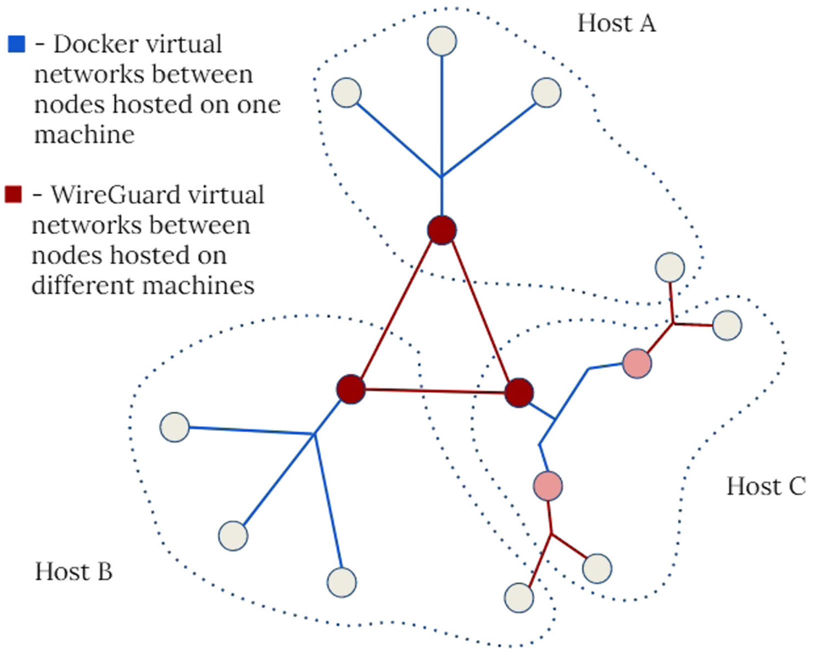

🟦 - Docker virtual networks between nodes hosted on one machine
🟥 - WireGuard virtual networks between nodes hosted on different machines
Host A
Host B
Host C

Fig. 3. Modeling a distributed network using the WireGuard tool.

2. "Laboratory" starts by creating network nodes based on Docker containers and network connections between them.
3. Displaying the emulated network is available in the web interface for conducting experiments and monitoring their progress.

6 Implementations of the Proposed Architecture in L3NS and MADT

We have tested several approaches to the implementation of the proposed architecture. The first of them is the MADT program complex, divided into three parts: a library for describing "Laboratories", a server for launching them, and a monitoring interface with zmq sockets. The second approach is implemented in the form of the L3NS library containing all the components necessary for modelling the network in the format of one programmable solution. While MADT allows to familiarize yourself quickly with the concept of the proposed architecture and set the experiment visually, L3NS has the functionality to more quickly and flexibly describe the model of node interaction and implementation in the CI ecosystem.

7 Example: Modeling in Applications with Client-Server Architecture

Let us consider how, using the proposed concept. A network is built with a set of routers and dynamic routing. Such a system can be represented as a tree, the nodes of which are routers and clients. Figure 4 shows how the network looks in case of height of the tree, and the number of leaves is set by 3 and 3.

The construction of such a network is carried out according to the following principle: the graph is expanded using several iterations of adding subnets for child nodes. At the end of each iteration, the array of external nodes is updated. For all nodes, except the external ones, the operation of the dynamic routing protocol is configured. This allows traffic to pass through the tree.

Each child node of the graph is assigned a Docker image that is used to create the client node.

Another step that has to be taken for the correct operation of our virtual network is to define the server address for each client so that they can interact.

When the model starts, it becomes possible to set network problems manually and monitor how they affect the application [21].

8 Modeling Management in the MADT Visual Interface

In MADT, after saving the configuration of the built network, we can go to the graphical interface and monitor the operation of the application. To configure the network status, you have to click on any node and set the interference parameters using the window that opens (Fig. 5a) and then press the "tcset" button. For example, to simulate 50% package loss for a router, we need to set the corresponding value of the loss field.

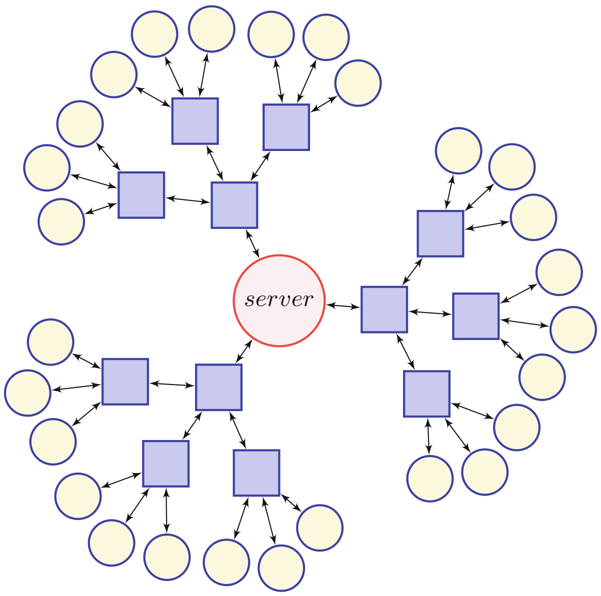
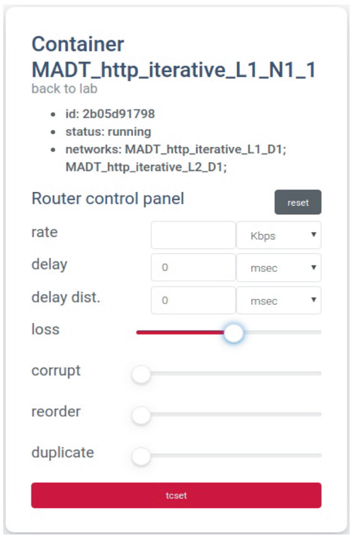


Fig. 4. Network model: server in the middle, blue boxes represent routers, yellow circles depict client nodes. (Color figure online)



Container MADT_http_iterative_L1_N1_1

back to lab

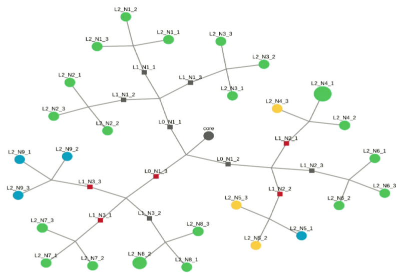
- id: 2b05d91798
- status: running
- networks: MADT_http_iterative_L1_D1; MADT_http_iterative_L2_D1;

Router control panel reset

rate		Kbps
delay	0	msec
delay dist.	0	msec
loss		
corrupt		
reorder		
duplicate		

tcset

(a) Setting parameters for a single node



(b) Visualization of emulated network

Fig. 5. MADT visual interface. (Color figure online)

After we set the problem for several nodes, we can observe the result on the graph. The colour of the node depends on the status of the incoming message. Green indicates regular operation, and yellow indicates a malfunction, red indicates critical problems, and purple indicates unexpected exceptions (Fig. 5b).

9 Differences in Modelling Capabilities of the L3NS Library and the MADT Software Package

The L3NS code is independent of MADT. The main functional features are the logic of the distribution of IP-addresses. Addresses are distributed according to the description of the "Laboratory", due to which the IP addresses of some nodes can be used to describe others (for example, immediately define the server address for clients) [10]. Also, work with Docker networks has been dramatically simplified in L3NS: network IP addresses, as well as the IP addresses of nodes in them, are set directly, while in MADT a second IP address is given to each interface on the network.

When describing network elements, different strategies for starting and stopping nodes can be used in L3NS. For example, using the local Docker container submodule, the nodes are launched as Docker containers. Meanwhile using cluster submodule, the containers are started on remote machines. L3NS provides the ability to configure remote machines via ssh. In other words, it is possible to access the client on a specific host, and, for example, view a list of Docker images used on it. To run the service system model on several virtual machines, you need to:

- Configure ssh keys,
- Install Docker,
- Install dependencies in python,
- Install WireGuard.

In L3NS support for parallel launching of containers is implemented using ThreadPoolExecutor, which speeds up the launch of "Laboratories" by order of magnitude, as well as networking via Docker Swarm [23] compared to MADT. The approach based on the creation of a single solution in the form of a library for a user programmer showed greater flexibility in system scaling tasks than the MADT program package, which separates the functions of describing, running and configuring "Laboratories".

10 Conclusion

The solutions we developed for testing distributed service systems allow to:

- Create realistic models of large IP networks;
- Deploy distributed applications based on Docker images;
- Manage the quality of work of different parts of the simulated network;

– Visualize the state of a distributed application working in real-time.

Two approaches were implemented — the MADT software package and the self-contained L3NS library. Using the library, it is convenient to test network interaction by emulating the network both on one node and simulating the interaction of nodes in a distributed model. Its use can significantly accelerate the launch of a simulated network without a web interface. While using the complex of programs MADT, it is easy to carry out interactive search experiments. Thus, MADT and L3NS can be used for:

– Comparing the work of several distributed applications under the same conditions;
– Checking application stability regarding unstable network operation;
– Studying the impact of network structure on the operation of each application;
– Checking for network plan vulnerabilities in application interaction.

Acknowledgments. This research was partially supported by the Russian Foundation for Basic Research grant (project no. 17-29-04288).

References

1. EVE-NG webpage. https://www.eve-ng.net/. Last Accessed 20 June 2020
2. GNS3 webpage. https://www.gns3.com/. Last Accessed 20 June 2020
3. Gremlin webpage. https://www.gremlin.com/. Last Accessed 20 June 2020
4. L3NS GitHub repository. https://github.com/rukmarr/l3ns. Last Accessed 20 June 2020
5. MADT GitHub repository. https://github.com/dltcspbu/madt. Last Accessed 20 June 2020
6. Pumba GitHub repository. https://github.com/alexei-led/pumba. Last Accessed 20 June 2020
7. Al-Rakhami, M., et al.: A lightweight and cost effective edge intelligence architecture based on containerization technology. World Wide Web **23**, 1–20 (2019)
8. Barrachina-Muñoz, S., Wilhelmi, F., Selinis, I., Bellalta, B.: Komondor: A wireless network simulator for next-generation high-density wlans. In: 2019 Wireless Days (WD), pp. 1–8. IEEE (2019)
9. Bhushan, K., Gupta, B.B.: Distributed denial of service (ddos) attack mitigation in software defined network (sdn)-based cloud computing environment. J. Ambient Intell. Hum. Comput. **10**(5), 1985–1997 (2019)
10. Huo, C., Yuan, J., Song, G., Shi, Z.: Node reliability based multi-path routing algorithm of high-speed power line communication network. In: 2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), pp. 570–573. IEEE (2019)
11. Iakushkin, O., Malevanniy, D., Bogdanov, A., Sedova, O.: Adaptation and deployment of panda task management system for a private cloud infrastructure. pp. 438–447. Springer International Publishing (2017)
12. Koganty, R., Alex, N., Su, C.H.: Framework for networking and security services in virtual networks, US Patent 10203972, 12 Feb 2019

13. Lang, D., Jiang, H., Ding, W., Bai, Y.: Research on docker role access control mechanism based on drbac. In: Journal of Physics: Conference Series. vol. 1168, p. 032127. IOP Publishing (2019)
14. Li, T., Gopalan, K., Yang, P.: Containervisor: Customized control of container resources. In: 2019 IEEE International Conference on Cloud Engineering (IC2E), pp. 190–199. IEEE (2019)
15. Lipp, B., Blanchet, B., Bhargavan, K.: A mechanised cryptographic proof of the wireguard virtual private network protocol. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 231–246. IEEE (2019)
16. Lofstead, J., Baker, J., Younge, A.: Data pallets: containerizing storage for reproducibility and traceability. In: International Conference on High Performance Computing, pp. 36–45. Springer (2019)
17. Mahmud, R., Buyya, R.: Modelling and simulation of fog and edge computing environments using ifogsim toolkit. In: Buyya, R., Srirama, S.N. (eds.) Fog and Edge Computing: Principles and Paradigms, pp. 1–35. Wiley, Hoboken (2019)
18. Malevanniy, D., Iakushkin, O., Korkhov, V.: Simulation of distributed applications based on containerization technology. In: Computational Science and Its Applications - ICCSA 2019, pp. 587–595. Springer International Publishing (2019)
19. Malevanniy, D., Sedova, O., Iakushkin, O.: Controlled remote usage of private shared resources via docker and novnc. In: Computational Science and Its Applications - ICCSA 2019, pp. 782–791. Springer International Publishing (2019)
20. Odun-Ayo, I., Geteloma, V., Eweoya, I., Ahuja, R.: Virtualization, containerization, composition, and orchestration of cloud computing services. In: International Conference on Computational Science and Its Applications, pp. 403–417. Springer (2019)
21. Paillisse, J., Subira, J., Lopez, A., Rodriguez-Natal, A., Ermagan, V., Maino, F., Cabellos, A.: Distributed access control with blockchain. In: ICC 2019–2019 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2019)
22. Sedayao, J.C., Smith, C.A., Li, H., Yoshii, T.H., Black, C.D., Hassan, V., Stone, D.W.: Virtual core abstraction for cloud computing, US Patent 10176018, 8 Jan 2019
23. Shah, J., Dubaria, D.: Building modern clouds: using docker, kubernetes & google cloud platform. In: 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), pp. 0184–0189. IEEE (2019)
24. Son, J., Buyya, R.: Latency-aware virtualized network function provisioning for distributed edge clouds. J. Syst. Softw. **152**, 24–31 (2019)
25. Tarasov, V., Rupprecht, L., Skourtis, D., Li, W., Rangaswami, R., Zhao, M.: Evaluating docker storage performance: from workloads to graph drivers. Clust. Comput. **22**(4), 1159–1172 (2019)
26. Zaman, F.A., Jarray, A., Karmouch, A.: Software defined network-based edge cloud resource allocation framework. IEEE Access **7**, 10672–10690 (2019)