MACE: A Mobile Ad-hoc Computing Emulation Framework

Bruno Chianca Ferreira, Guillaume Dufour, Guthemberg Silvestre

MACE: A Mobile Ad-hoc Computing Emulation Framework

Bruno Chianca Ferreira[1], Guillaume Dufour[2], and Guthemberg Silvestre[3]

[1] ENAC, Université de Toulouse, France
bruno.chianca@enac.fr
[2] ONERA, The French Aerospace Lab
guillaume.dufour@onera.fr
[3] ENAC, Université de Toulouse, France
silvestre@enac.fr

Abstract. Designing and deploying mobile applications for Edge/Fog environments is challenging. Previous studies have introduced simulation and emulation tools that can help to model new applications but fail to deliver an environment in which a system designer can emulate a realistic deployment of such applications. We propose the Mobile Ad-hoc Computing Framework (MACE)[4] that provides features which facilitate and accelerate the development and testing of Edge/Fog applications. It also provides the nodes with mobility control that can be modelled or triggered by external applications, such as a flight simulator. With the included GUI, the user can plan deployment topologies, observe and interact with the nodes in real-time. To demonstrate its functionalities, two experimental evaluations were performed: one comparing the network emulators and another with an *off-the-shelf* configuration service running in mobile nodes. Finally, the functionality of controlling mobility via an external application is also demonstrated.

Keywords: fog, edge, iot, emulator, framework, mobile, ad-hoc

1 Introduction

The research field in distributed systems has witnessed a recent growing interest in mobile distributed computing, especially in the areas of Mobile Cloud Computing (MCC) [1], Mobile Edge Computing (MEC) [2] and Mobile Ad-Hoc Computing (MAC) [3]. This growth was principally driven by the adoption of IoT, Edge and Fog applications and the market demand for low latency in the edge. To support this research, many simulators were created [4], however, there is a lack of emulation tools for mobile Edge/Fog computing. Some important features are missing, such as full-scale emulated deployment where prototypes coexist with off-the-shelf applications or more options of mobility control. Previously, containers and virtual machines together with software defined networks

[4] https://github.com/brunobcfum/pymace

(SDN) emulators have enabled system designers to run several nodes of distributed computers by emulating the topology of a wired network. However, when the application focuses on nodes running on a Mobile Ad-Hoc Network (MANET) or a Flying Ad-Hoc Network (FANET), difficulties arise with the mobility and the representation of nodes' unstable connectivity. This study proposes a framework that enables the emulation of mobile distributed applications in a virtual environment, so that system designers can easily modify scenarios and topologies composed by mobile wireless nodes.

1.1 Motivation

Even though simulation platforms are very useful in early-stage development, emulation introduces the possibility of running software in the late development stage, so that an emulated node can be easily substituted by a physical device. Most of the simulators require platform-specific development with custom libraries and have as goal creating detailed models of each layer of a network stack. In this work, the focus is not on studying the network stack itself, but distributed applications, which eliminates the need of simulating detailed network behaviour.

The main motivation behind this work was the lack of a wireless emulation software which allows designing and testing mobile distributed algorithms and software, such as distributed coordination services, distributed data stores, task scheduling and offloading. When designing a mobile distributed system connected via wireless interfaces, new requirements arise that cannot be fulfilled by existing tools. Section 1.2 illustrates available solutions that also intended for IoT, Edge and Fog environments. The main features that cannot be altogether fulfilled by any existing tool are: support of mobile wireless nodes through emulation; flexibility in the creation of new topologies such as positioning of the nodes and mobility models; scalability that allows both a higher number of nodes and high network traffic; possibility of running applications directly in the host, in virtual machines or containers and the ability to emulate constrained nodes; capacity to be integrated with external software for mobility control.

1.2 State of the Art

This section presents a review of prior work relevant in the context of emulation for Edge/Fog environments, highlighting why they cannot fully meet the requirements for fast, easy-to-instrument mobile distributed application development. Table 1 shows what is, to the best of our knowledge, the current state of the art and illustrates the features it does not cover.

FogNetSim++ [5] is a simulator built with OMNet++ that proposes an open-source toolkit for modeling new fog applications. This simulator fails as many others to allow that simulated applications can be directly deployed to devices without the need for adaptation to an environment outside the OMNet++ framework. In [6] the author expands the usability of FogNetSim++ by introducing a named pipes communication channel between the simulator and the application

Table 1: Emulators Comparison

Functionality / Emulator	Network Emulator	Mobility Control	Distributed Mobile Applications					Energy Model	Resource Management & Schedulling	Open Source	Last Commit
			Linux Native Applications	ARM VMs	x86 VMs	x86 Containers	Embedded Applications				
FogNetSim++ [5]	OMNet++	✓						✓	✓	✓	3y ago
FogNetSim++ w/ NodeRed[6]	OMNet++	✓					NodeRed	✓	✓		
iFogSim [7, 8]	Built-in	✓						✓	✓	✓	1y ago
Miniworld [9]	CORE			Possibly	✓	✓				✓	4y ago
Virtual Mesh [10, 11]	OMNet++	✓	✓					✓			
MockFog [12]	Cloud Infra			Possibly	✓	✓			Since v.2	✓	7m ago
EmuFog [13]	MaxiNet					✓			✓	✓	3m ago
EmuEdge [14]	Linux veths	✓		Android	✓	✓				✓	3y ago
MACE	CORE	✓	✓	✓	✓	✓	RiotOS			✓	1m ago

NodeRed[5]. The concept idealized by the author is similar to one of the goals of this work: code once, simulate and deploy. However, MACE aims in going a bit further by providing the option of running not only applications aimed in IoT devices, but also VMs and containers that can run in edge clouds, or cloudlets. iFogSim is a modular toolkit proposed in [7] that has key features for Fog simulations such as power monitoring and resource management and is initially based on the well-known cloud simulator CloudSim[15]. In [8], the author extended the work of iFogSim to add a data placement extension with the goal of enabling users to experiment with different policies. Miniworld [9] uses CORE as the network emulator and provides the possibility of deploying virtual machines and containers as emulated nodes. However, it didn't include mobility models or the integration with other simulators. Another missing feature was the possibility of running constrained nodes with QEMU, embedded operating systems, host applications, and containers. Virtual Mesh [10, 11] had the goal of enabling users to test their mesh network projects in an emulated environment. The user would be able to create applications to run directly in the host operating systems, whilst the network interfaces used by the application would be emulated in OMNet++. MockFog [12] proposes an emulation environment deployed in existing cloud infrastructure providers to allow the configuration of an heterogeneous environment. However, it is not suited for emulation of mobile Ad-Hoc nodes due to the limitations in the topology definition and lack of mobility control, and also cannot be used to emulate constrained devices. EmuFog [13], like this work, uses a network emulator in order to allow the deployment of several nodes in a controlled environment. The emulator used is MaxiNet, and the applications are emulated in a Docker container as can be done in this work but lack the option of adding mobility to the nodes. EmuEdge [14] is an emulator

[5] https://nodered.org/about/

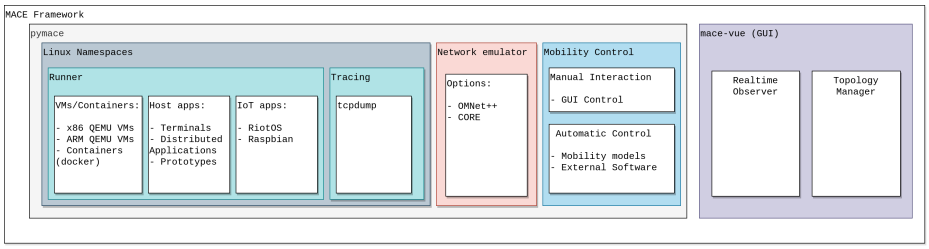


Fig. 1: Framework Architecture

with similar goals of creating realistic experiments. It uses *Xen* to create virtual nodes and *SDN* for creating and managing the topologies. It allows also detailed network tracing and also replaying traffic generated by network simulators, but lacks good wireless and mobility implementations. It also lacks the possibility of emulating constrained IoT nodes, focusing on Linux and Android VM nodes. Another possibility of testing distributed prototypes in a realistic environment is to use test beds such as IoTLab [16], which allow the download of applications on physical embedded hardware and the observation of realistic radio behaviour.

1.3 Contribution

The main contribution of this study was to establish a framework that enables researchers and engineers to work with all the aforementioned features in a simplified way. It not only enables the study and evaluation of distributed mobile applications, but also allows faster prototyping since it does not require additional software specific skills for designing applications. It has a GUI to facilitate the management and supervision of scenarios and topologies composed of nodes running as virtual machines, containers or applications running directly in the host OS. The framework was also designed with the integration possibility of other external tools for mobility control. The source code of this framework is available at: *https://github.com/brunobcfum/pymace* [6]. A demonstration of a mobile system running an etcd [17] configuration service is presented as an experimental evaluation in Section 3.2.

2 Proposed Framework

The proposed mobile emulation framework creates an environment composed of different modules that can be configured according to the need and Figure 1 illustrates the main elements.

[6] This software is constant evolution, and new features might be introduced in the future that are not described in this paper.

2.1 Network Emulator

Network emulators, unlike the network simulators, aim in mimicking the behaviour of the device without modelling in detail each intrinsic aspect. The replacement of an emulated network interface by a physical one is imperceptible by the applications. Such emulators allow the system designers to experience a more realistic approach for the tests. They provide for the application the perception that it is running on real hardware. For MACE two different network emulators were considered, tested and are provided as options: OMNet++ together with the Emulation capability supplied by the INET Framework and CORE. When running an emulation scenario, the user can choose between one or another by just changing a configuration.

OMNet++ is an open-source software written predominantly in C++ that is widely used in the academic world and has good acceptance in wireless simulation studies. OMNet++ is not intended to be a simulator, but a framework mainly used to build network simulators. Therefore, it features several libraries that can be coupled together to build a stack which once compiled, operates as a simulator. Additional elements can be added to OMNet++ through third part frameworks. The most important framework is INET, which among other things allows the creation of emulated network interfaces when using a real-time scheduler. When using the OMNet++/INET emulation option, the network stack is split so that part of the stack is emulated in the operating system itself by creating virtual network interfaces, and part is simulated in OMNet++. The open-source project CORE works similarly to OMNet++ when it is running emulated interfaces, but the depiction of the wireless interface is less customizable. Additionally, instead of using TAP interfaces it uses VETH interfaces, which despite being a different technology, from the perspective of an emulated node achieves the same results. It uses *ebtables* to control the connectivity in the physical medium similar to the Unit Disk Radio module from OMNet++. Which means that the signal attenuation is not realistically represented, so the connectivity is defined by the Cartesian distance between the nodes. When there is a need of emulating a more realistic representation of the wireless interface, taking in consideration propagation delays and signal decay due to the specific medium characteristics, the user can associate the emulator EMANE [18]. EMANE (Extendable Mobile Ad-hoc Network Emulator), is focused in mobile Ad-hoc networks and has different radio models available such as IEEE802.11abg, TDMA, LTE and Radio Pipe.

2.2 Routing Protocol

Another important component of the framework is the routing protocol to enable the multi-hop communication between the mobile nodes. Many different protocols have been created and each one of those has an advantage or specific goal depending on the application [19, 20]. In this work the interest in upon a routing protocol implemented for Linux, which can be used in generic applications and work well with mobile nodes. A comparative study [21] has been made with some

of the most prominent protocols for MANETs and FANETs: AODV [22], OLSR [23] and BATMAN [24]. BATMAN V was found to have better packet delivery ratio with mobility and was capable of maintaining good connectivity even with high mobility. It is also a proactive protocol which can reduce latency since it constantly updates the best route among the nodes.

2.3 Mobility Control

The mobility module of the framework is a software capable of controlling the position of each node in the simulation via manual input or via automatic control. The automatic control can happen via the implementation of mobility models or via an interface to external software. The positions are fed into the network emulator that, based in configurations of the wireless interface's radio reach, connects each node taking part in the distributed application. When using CORE as network emulator, mobility is quite limited. By default, the only option to add mobility is to create predefined scripts associated with each node. They, therefore, loop through this script creating a repetitive movement. Nodes can be initially placed with the help of a graphical interface supplied with CORE. To avoid this limitation, it is integrated into the proposed framework the option of using a Python library[7] that can be used to control the mobility of the nodes and their position injected into the CORE emulation session. OMNet++, on the other hand, comes with several mobility models provided by the INET framework, so the user only needs to configure the emulation session according to the pattern more suited to the application. Due to the proposed framework's modularity, other types of domain-specific simulators can potentially be used to control the nodes' mobility. A proof-of-concept experiment is later introduced in Section 3 where a UAV Flight simulator is used to updated nodes' positions in CORE. Other works have also introduced similar functions with OMNet++ [25][26].

2.4 Distributed Mobile Applications

Running emulation scenarios instead of simulated applications on simulation frameworks create opportunities to run distributed mobile applications in diverse scenarios. That consequently increases the ability to create heterogeneous sessions by running some nodes directly on the host, and others inside virtual machines or containers as shown in Figure 2. Each node can be tuned with different hardware definitions such as available persistent and volatile memory or the number of processor cores. Furthermore, it is also possible to mix nodes running in different architectures such as x86 and ARM. The same is true when running native applications together with others made for different operating systems such as RiotOS[27], or other embedded OSes that can be compiled as a host application for testing.

[7] https://github.com/panisson/pymobility

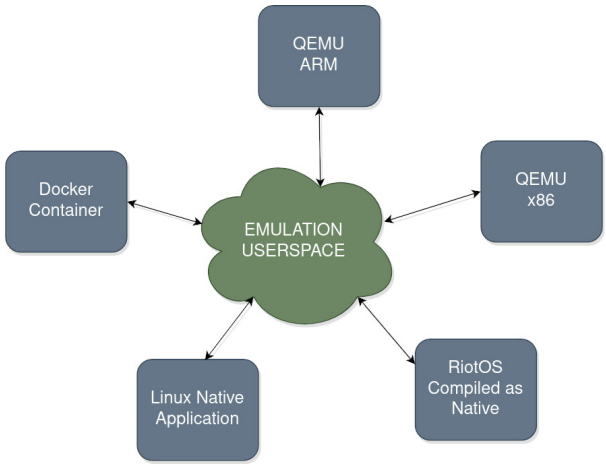


Fig. 2: Emulation Options

Linux Native Applications When running applications inside Linux namespaces, due to the isolation, they can open sockets using the same port number. Hence, it is possible to run any type of application that accept more than one simultaneous instance of the process. That allows for example testing several instances of web servers or distributed file systems. An evaluation environment was created to run a mobile distributed experiment with etcd and more details can be found in Section 3.2. When it is necessary to increase the level of isolation, the application can be run as a VM or a container.

Running Virtual Machine Nodes Instead of running distributed applications directly in the host operating system, it is also possible to run guest virtual machines (VM). The framework provides the option of running a VM or a Docker container in each node where the virtual network interfaces will bridged to the emulated one. The framework provide a simple way of managing the preconfigure VMs images which can be x86 or ARM for embedded devices. The overall architecture can be observed in Figure 3.

Running Embedded Operating Systems One important step of developing distributed algorithms for heterogeneous IoT devices is the ability to test them on different operating systems and hardware with unbalanced performance. Nowadays, many embedded OSes are available and RiotOS for instance, provides the option of compiling the firmware as a x86 (native) application for testing. With that, one can test the communication between the emulated firmware and other nodes in the system, such as an edge server running on a virtual machine in another node.

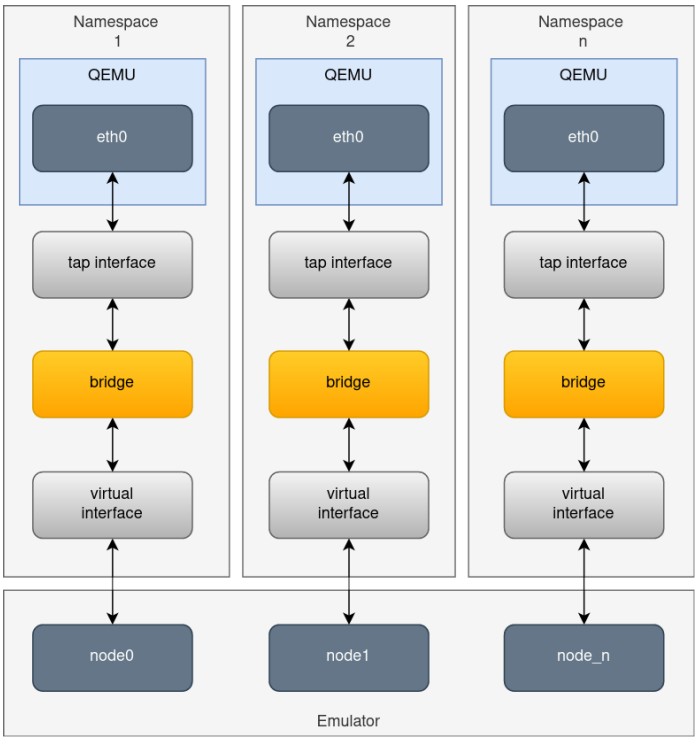


Fig. 3: Virtual Machine Architecture

3 Experimental Evaluation

Since the network emulator is an important component of the framework, some tests were run to assess the maximum performance that can be expected from them and build a baseline for further assessments. For that, benchmarks were taken with *iperf3*[28]. The reasoning was to inject different traffic flows in a topology and measure the throughput and losses. All the tests were performed in the computer described in Table 2. The experimented topology is composed of nine nodes symmetrically displaced and with an equidistant one-hop distance between each other. Figure 4 illustrates the topology. An emulated wireless Ad-Hoc network with the parameters set as illustrated in Table 3 was created in OMNet++ and in CORE.

3.1 Throughput

For measuring the throughput, the worse case was considered. Flows were created traversing the maximum distance available in the topology, between the nodes in the diagonal extreme corners. When using OMNet++, the real-time

Table 2: Test Platform

Processor Manufacturer / Model:	AMD / Ryzen 7 1700
Number of cores	8 (16 Threads)
Cache L1,L2,L3	768kB, 4MB, 16MB
Memory	16GB DDR4
Emulator	OMNet++ 5.6.1 and CORE 7.2.1
Operating System	Linux Mint 20 x86 64bits
Routing Protocol	B.A.T.M.A.N V[24]

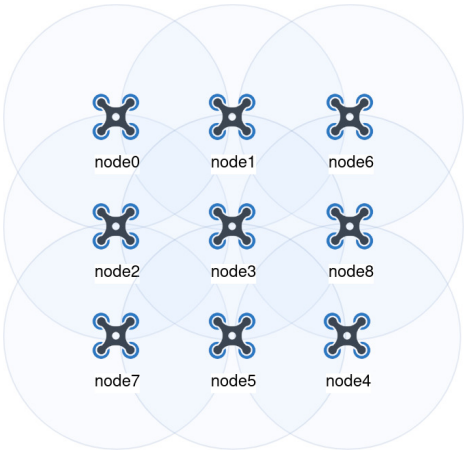


Fig. 4: Symmetrical topology

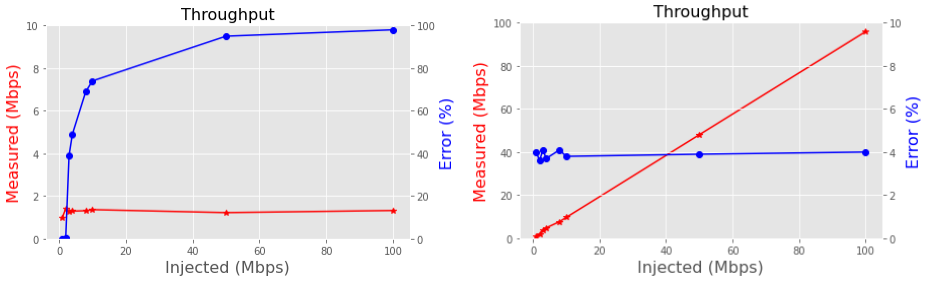
scheduler needs to keep up with processing all the packets in the whole topology. Furthermore, the scheduler must run in only one thread, and as seen in Figure 5(a), when there are too many packets it cannot keep up. Even when injecting throughput below the theoretical expected based in the configuration, it can deliver less than 2Mbps. So, all injected flow beyond this threshold is ultimately dropped, with losses reaching almost 100% when injecting 100Mbps. CORE on the other hand, could keep up to the injected flow whilst keeping the error rate fixed in the configured value of 1% per hop as can be seen in Figure 5(b).

Table 4: etcd Parameters

Key size	8 Bytes
Value size	256 Bytes
etcd version	3.4.13
Fixed leader	Node 3

Table 3: Emulation parameters

Setting	OMNet++	CORE
MTU	1500B	1500B
Interface	ExtUpperIeee80211Interface	N/A
MAC	Ieee80211Mac	N/A
Management	Ieee80211MgmtAdhoc	N/A
Op Mode	AC	N/A
Bitrate	433.3Mbps	433.3Mbps
Number of antennas	6	N/A
Delay	N/A	1300us
Jitter	N/A	5 us
Error rate	N/A	1



(a) OMNet++ (b) CORE

Fig. 5: Measured throughput traversing the topology

3.2 Distributed Mobile Applications with etcd

etcd [17] is a distributed key-value store which has a broad acceptance among system designers and is by tools largely used in production such as a configuration service for Kubernetes. It provides strong consistency among the nodes by using Raft as consensus protocol [29] and can be used by any application that requires consistent key-value store. For this proof of concept, the proposed framework is used to create an emulation session where etcd is deployed in namespaces using ramdisks as storage units. Etcd has available in their website baseline performance figures[8], alongside a benchmark tool that can run in any deployed system. Since the traffic flow when testing etcd is too high, only CORE could be used and the testing platform and settings are the same already shown in Table 2 with the topology presented in Figure 4. The tests were performed with the same parameters as stated on their website and shown in Table 4.

It is possible to see that the results are below the baseline, which is expected considering the high latency configured in CORE for the links. Reducing the latency to 300us instead of 1300us increased the average queries per second to

[8] https://etcd.io/docs/v3.4.0/op-guide/performance/

Table 5: etcd Results

1 con. 1 cli.	Baseline	1300us	300us	300us mob.
Average QPS	583	231	567	40
Average Latency	1.6ms	43ms	18ms	24ms
100 con. 1000 cli.	**Baseline**	**1300us**	**300us**	**300us mob.**
Average QPS	44 341	7 184	11 172	4432
Average Latency	22ms	137ms	85ms	224ms

11172, and the average latency was reduced to 85ms. Mobility was then added with the random walk model provided by a third part library. As seen in Table 5, with mobility there is a considerable decrease in performance, with lower throughput and higher latency. The mobility can also be controlled by an external agent related to the specific application domain. To test this, the emulator was connected to an open-source UAV flight simulator. Paparazzi[30] is an autopilot developed for fixed and rotary wing UAVs, and when using Paparazzi, all the UAVs are controlled by the ground station via radio commands. However, Paparazzi is also suited with a flight simulator where the radio link between the UAVs and the ground station is replaced by a UDP sockets communicating via *pprzlink*[9]. MACE also includes a proxy for the pprzlink that can capture all packets exchanged between the simulated UAVs and the ground station. As a result, the emulator can capture in real-time, the simulated GPS position of the UAVs and update the emulated topology.

4 Future work and Limitations

One of the main limitations in this work were the issues when running OMNet++ as the network emulator. When increasing the number of packets, the emulator cannot keep up with the traffic even though the computer processor usage doesn't reach it's limit. That could indicate that it's code can be potentially enhanced to achieve better results. Another observation was that when using CORE, the BATMAN routing protocol requires more than 20 seconds for the routes to converge, so that traffic can start flowing. When using OMNet++ on the other hand, it requires less than 2 seconds. In future improvements of the MACE framework, new functions useful when experimenting with Edge/Fog infrastructure are still to be added: cost model, automatic application migration and energy model. Also, to increase scalability, we plan to explore CORE's feature of running distributed emulation sessions.

5 Conclusions

This work introduced the MACE emulation framework, which is aimed on Edge/-Fog application development and with particular focus on Mobile Ad-hoc Com-

[9] A communication protocol developed for serializing Paparazzi communication.

puting. It is built on top of network emulators, delivering functionalities that enable fast application prototyping, instrumentation and emulated deployment. The main functionalities were demonstrated by experimenting with an off-the-shelf distributed configuration service, and showed how an emulated environment can be deployed in order to enable reproducible research with mobile distributed computing. The source code of this framework is available at: *https://github.com/brunobcfum/pymace*.

References

1. H. Qi and A. Gani, "Research on mobile cloud computing: Review, trend and perspectives," *2012 2nd International Conference on Digital Information and Communication Technology and its Applications, DICTAP 2012*, pp. 195–202, 2012.
2. H. Elazhary, "Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions," *Journal of Network and Computer Applications*, vol. 128, no. November 2018, pp. 105–140, 2019. [Online]. Available: https://doi.org/10.1016/j.jnca.2018.10.021
3. I. Yaqoob, E. Ahmed, A. Gani, S. Mokhtar, M. Imran, and S. Guizani, "Mobile ad hoc cloud: A survey," *Wireless Communications and Mobile Computing*, 2016.
4. A. Markus and A. Kertesz, "A survey and taxonomy of simulation environments modelling fog computing," *Simulation Modelling Practice and Theory*, vol. 101, p. 102042, 2020, modeling and Simulation of Fog Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X1930173X
5. T. Qayyum, A. W. Malik, M. A. Khan Khattak, O. Khalid, and S. U. Khan, "Fognetsim++: A toolkit for modeling and simulation of distributed fog environment," *IEEE Access*, vol. 6, pp. 63 570–63 583, 2018.
6. F. Fam, D. F. S. Santos, and A. Perkusich, "Integrating an IoT Application Middleware with a Fog and Edge Computing Simulator," 2020.
7. H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2509
8. M. I. Naas, J. Boukhobza, P. Raipin Parvedy, and L. Lemarchand, "An extension to ifogsim to enable the design of data placement strategies," in *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, 2018, pp. 1–8.
9. N. Schmidt, L. Baumgärtner, P. Lampe, K. Geihs, and B. Freisleben, "Miniworld: Resource-aware distributed network emulation via full virtualization," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, Jul. 2017, pp. 818–825.
10. R. Gantenbein, "Virtual Mesh : An Emulation Framework for Wireless Mesh Networks in OMNet++," *Master Thesis of the Philosophical-scientific Faculty of the University of Bern*, 2010.
11. T. Staub, R. Gantenbein, and T. Braun, "VirtualMesh: An emulation framework for wireless mesh and ad hoc networks in OMNeT++," *Simulation*, vol. 87, no. 1-2, pp. 66–81, 2011.

12. J. Hasenburg, M. Grambow, E. Grünewald, S. Huk, and D. Bermbach, "Mockfog: Emulating fog computing infrastructure in the cloud," in *2019 IEEE International Conference on Fog Computing (ICFC)*, 2019, pp. 144–152.
13. R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran, "Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures," in *2017 IEEE Fog World Congress (FWC)*, 2017, pp. 1–6.
14. Y. Zeng, M. Chao, and R. Stoleru, "Emuedge: A hybrid emulator for reproducible and realistic edge computing experiments," in *2019 IEEE International Conference on Fog Computing (ICFC)*, 2019, pp. 153–164.
15. R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, p. 23–50, Jan. 2011. [Online]. Available: https://doi.org/10.1002/spe.995
16. C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "Fit iot-lab: A large scale open experimental iot testbed," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, pp. 459–464.
17. *etcd*, 2020 (accessed October, 2020). [Online]. Available: https://etcd.io/
18. *EMANE Emulator*, 2020 (accessed October, 2020). [Online]. Available: https://github.com/adjacentlink/emane/wiki
19. O. S. Oubbati, M. Atiquzzaman, P. Lorenz, M. H. Tareque, and M. S. Hossain, "Routing in flying ad hoc networks: Survey, constraints, and future challenge perspectives," *IEEE Access*, vol. 7, pp. 81 057–81 105, 2019.
20. J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 6–28, 2004.
21. B. Sliwa, S. Falten, and C. Wietfeld, "Performance evaluation and optimization of b.a.t.m.a.n. v routing for aerial and ground-based mobile ad-hoc networks," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, 2019, pp. 1–7.
22. C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, 1999, pp. 90–100.
23. P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," in *Proceedings. IEEE International Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century.*, 2001, pp. 62–68.
24. *B.A.T.M.A.N*, 2020 (accessed October, 2020). [Online]. Available: https://www.open-mesh.org/projects/batman-adv/wiki
25. E. A. Marconato, M. Rodrigues, R. M. Pires, D. F. Pigatto, L. C. Q. Filho, A. S. R. Pinto, and K. R. L. J. C. Branco, "AVENS – A Novel Flying Ad Hoc Network Simulator with Automatic Code Generation for Unmanned Aircraft System," pp. 6275–6284, 2017.
26. *Veins The open source vehicular network simulation framework.*, (accessed October, 2020). [Online]. Available: https://veins.car2x.org/
27. *RiotOS*, 2020 (accessed October, 2020). [Online]. Available: https://riot-os.org/
28. *iPerf*, 2020 (accessed October, 2020). [Online]. Available: https://www.iperf.fr-en
29. L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and M. Kihl, "Impact of etcd deployment on kubernetes, istio, and application performance," *Software: Practice and Experience*, vol. 50, no. 10, pp. 1986–2007, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2885

30. G. Hattenberger, M. Bronz, and M. Gorraz, "Using the Paparazzi UAV System for Scientific Research," in *IMAV 2014, International Micro Air Vehicle Conference and Competition 2014*, Delft, Netherlands, Aug. 2014, pp. pp 247–252. [Online]. Available: https://hal-enac.archives-ouvertes.fr/hal-01059642