



HAL
open science

Fidelity-aware Large-scale Distributed Network Emulation

Houssam Elbouanani, Chadi Barakat, Walid Dabbous, Thierry Turletti

► **To cite this version:**

Houssam Elbouanani, Chadi Barakat, Walid Dabbous, Thierry Turletti. Fidelity-aware Large-scale Distributed Network Emulation. *Computer Networks*, 2024, 10.1016/j.comnet.2024.110531 . hal-04591699

HAL Id: hal-04591699

<https://inria.hal.science/hal-04591699v1>

Submitted on 29 May 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Fidelity-aware Large-scale Distributed Network Emulation

Houssam ElBouanani, Chadi Barakat, Walid Dabbous, Thierry Turletti
Inria, Université Côte d’Azur, France

Abstract—The design and development of new network protocols, architectures, and technologies requires an evaluation phase where the researcher must provide empirical evidence for the performance of their contributions, potentially in comparison to existing solutions. In this context, network emulation has proven to be an attractive approach as it offers more flexibility compared to traditional testing platforms, and more realism compared to simulation. Network emulators provide contained, customisable, and scalable testing environments both for researchers to evaluate their contributions and for the community to reproduce their results. However, two limitations to network emulation have been identified and well documented in the literature: its scalability limits and its accuracy issues. This paper documents our attempts to address these concerns. Our findings are distilled into Hifinet: a lightweight scalable and fidelity-aware distributed network emulator. We particularly show how Hifinet outperforms its state-of-the-art counterparts in terms of scalability and efficiency by working around the flaws of their design principles and the technological limitations of the tools they rely on. Hifinet is also fidelity-enhanced, in that it implements a well-theorised fidelity monitoring framework, which passively monitors emulated packet delays to evaluate realism of network emulation and accuracy of results. Another asset of Hifinet is its ability to infer underlying causes in case of erroneous emulation and guide the user through fixing them. This is achieved by using delay tomography algorithms and heuristics.

Index Terms—network emulation, emulation fidelity, passive measurement, delay tomography

I. INTRODUCTION

Network emulation refers to the simulation –using modern system and network virtualisation technologies– of a network environment in which multiple interconnected computer systems can interact in real-time as they would in a real-world scenario. It is a critical tool for the design, testing, and evaluation of communication systems, protocols, and applications. The goal of emulation is to provide a realistic representation of the network environment, including all its components, topologies, protocols, and traffic patterns. This helps researchers and engineers understand how these systems will behave in real-world scenarios and identify any potential issues before deployment. Mininet [1] is one such tool which offers a lightweight solution packaged with easy-to-use

abstractions. The use of virtualisation allows such network emulators to provide generic environments that can a diverse set of applications, by seamlessly running operating systems, network protocols, and software applications with little to no special adaptation.

Distributed network emulation provides several advantages over traditional network emulation tools. For example, it allows researchers to test their systems in a realistic, multi-node environment, and supports large-scale, complex networks that cannot be easily replicated in a single machine. It also provides fine-grained control over the network environment and allows for flexible configuration of different scenarios, making it ideal for a wide range of applications, including performance analysis, security testing, and development of new protocols and applications.

Despite its many benefits, distributed network emulation also faces several challenges, particularly scalability and accuracy. Studies [9], [10] have shown that such emulators tend to produce incorrect network behaviour under certain circumstances. Without a thorough analysis and investigation of the emulation, this can lead to biased experiment results and erroneous conclusions. Indeed, a network emulator’s fidelity, defined as the extent to which emulated networks operate behave as *real, hardware-based* ones, is directly impacted by the environment on which the tool runs: whether enough physical resources are available to accommodate virtual components, and how well these components are isolated from each other. Currently, a user who is concerned about accuracy has to evaluate the fidelity of their emulation by defining high-level key performance indicators appropriate to their scenario [8]. This fidelity monitoring step requires thorough analysis and modeling of the emulated scenario which adds complexity to experimentation and might discourage from using emulation altogether.

The general objective is thus to allow network academics and engineers to develop and to test network technologies in environments that more realistically resemble existing production networks in size and operation. To this end, we present in this paper a novel universal framework for monitoring the fidelity of single-machine and distributed network emulators in a scenario-agnostic manner. Its aim is to detect when results produced from emulation are not to be trusted, by looking at how well the finest and most precise network phenomenon –namely, the network delay– is emulated. In particular, we first attempt in Section II the design of a distributed network emulator (Bignet) that offers the most important features of its existing counterparts but adds performance and improves

This work was carried out with the support of the SLICES-SC project, funded by the European Union’s Horizon 2020 programme (grant 101008468). This work has received partial funding from the Fed4FIRE+ project under grant agreement No 732638 from the Horizon 2020 Research and Innovation Programme.

This journal paper is an extension of a shorter paper [6] previously presented in the TASIR (Testbeds for Advanced Systems Implementation and Research) workshop of Comsnets 2023 conference, and published in its proceedings.

scalability. In Section III we address accuracy issues by a conceptual and operational definition of fidelity and argue for packet delay as a universal metric that can help detect many causes of emulation inaccuracy, then present a framework that relies on the passive measurement of the emulated network delays to assess the fidelity of emulation-based experiments. We will introduce our framework’s methodology and design choices, then sketch an implementation within Bignet to finally propose Hifinet: a large-scale distributed network emulator powered with the fidelity monitoring framework.

II. LARGE-SCALE DISTRIBUTED NETWORK EMULATION

While emulation overcomes some limitations of both traditional testbeds and simulation methods, it does not offer a perfect, fault-free service. In particular, two problems that limit the performance of modern virtualisation- and containersation-based network emulators have been identified: scale and accuracy. In this section, we will examine how the former is not yet fully overcome in practice by distributed emulation. We will present the case of Distrinet [4], which is proven to be the most scalable Mininet-like emulator (others include Mininet Cluster Edition (CE) [2] and Maxinet [11]), and build a new lightweight distributed network emulator from our proposed solutions to Distrinet’s limitations.

1. State-of-the-art Distributed Network Emulation

a) Perfect Compatibility with Mininet: One of the key features of the aforementioned distributed emulators, emerging from their design requirement to be perfectly compatible with Mininet, is the implementation with the latter’s interface. The logical basis for this self-imposed constraint is to appeal to users who are already familiar with the original emulator’s interface, and not to burden them with learning a new interface. While this has been a noble ambition, the fact of the matter remains that Mininet’s interface is only suitable for a laptop setting and was never meant to work perfectly in multi-node environments. In particular, Mininet provides its users with a shell-like interface through which they can run commands in any emulated end-host or networking node they wish, in addition to new Mininet-specific commands for network management. In single-machine settings, this is straightforward as commands can be simply redirected to their destination’s network namespace. In distributed settings, however, the commands need to be sent to the destination containers for execution. Distrinet achieves this by setting up a *management network* that connects all running containers to the client through the master machine, and Maxinet by maintaining open SSH sessions with each container. This approach incurs significant cost in terms of:

- Number of open SSH connections: for each running container one SSH session needs to be maintained. This impacts scalability as Linux imposes a hard limit on the number of simultaneously open SSH connections;
- Number of open files: for each running container, multiple Linux files need to be kept open, both for the SSH session and for the shell-like interface. This also impacts scalability as Linux imposes limits on the number of open files.

b) Heavyweight Containers: A second design choice made by the creators of Distrinet is the use of a natively-supported, open-source containerisation solution: LXC (Linux Containers). This Linux-supported technology offers more isolation at the cost of more resource usage. In particular, LXC provides *system containers*, which are different from regular *application containers* in that the latter achieve the illusion of isolation by running applications (and the libraries on which they rely) on virtual environments that use the host machine’s Linux kernel. Application containers are additionally offered their own network stack, their own file subtree (rooted somewhere in the host’s tree), and their own share of the machine’s resources. On the other hand, system containers are closer to regular virtual machines, as each container runs its own operating system, while sharing only a minimal part of the host’s kernel. This makes the LXC containers heavyweight: they consume more, and need more time to be deployed and stopped.

c) Containerised Virtual Switches: Another important feature of Distrinet is the isolation of virtual switches inside their own containers. Unlike its ancestor and its competitors, Distrinet was designed with the choice to run virtual switches in isolated environments similar to the emulated end-hosts. This is unfortunately a double-edge sword: while it lets emulated networking nodes be isolated and thus specific amounts of resources be allocated, it incurs containerisation overhead and elongates the path to and from the switches’ ports.

2. Bignet: a Scalable Distributed Network Emulator

From the observed flaws of the currently available emulators we build Bignet: a new distributed network emulator capable of overcoming the discussed limitations. The remainder of this chapter will lay out its design principles and introduce the blueprints of a lightweight first implementation. The section will end with a comparison against Distrinet and demonstration of our solution’s performance.

a) Design and Implementation: Bignet’s main technological difference compared to Distrinet is its use of lighter containers. Although far from being the micro-containers used by Mininet to emulate end-hosts, the containers used by Bignet still consume as much resources (computing, memory, and storage) compared to Maxinet and less compared to Distrinet that relies on heavy containers and/or virtual machines. This important technological shift increases scalability both by optimising resource usage and by reducing the amount of time needed for starting and stopping the containers. These Bignet containers can be prepared beforehand as images, and can be tailored for the emulated scenario and for each individual host.

In particular, Bignet uses Docker [3] to create application containers that emulate end-hosts. This lets users prepare the end-host’s configuration and their software into Docker images which will be deployed to emulate the corresponding emulated hosts. As for virtual switches and other networking nodes, they are by default run directly on the physical machines to increase performance and scalability by reducing overhead, with the option to be containerised as in Distrinet if the user wishes for more isolation.

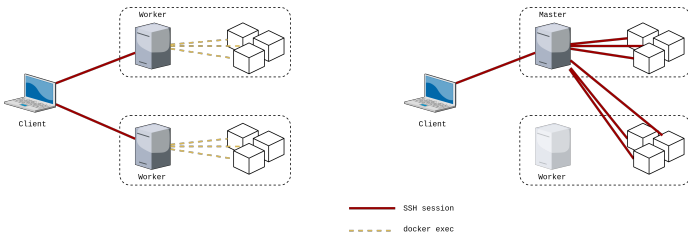


Fig. 1: Command execution in containers in Bignet (left) and Distrinet (right). In Bignet, workers act as gateways to their hosted containers via the Docker API (dashed yellow lines), which significantly reduces the number of open SSH sessions (red lines).

Another distinctive characteristic of Bignet compared to other emulators is the deviation from Mininet’s requirement for an interactive interface. Instead, Bignet focuses on a scripted scenario API: the user writes in great details the flow of events in scripts that the emulator will run non-interactively. By breaking off from this restrictive assumption, Bignet can be implemented without the need for a shell environment and live connections to the containers (as is the case for Distrinet, Mininet CE and Maxinet) –which would need to stay open during the emulation regardless of whether or not commands are sent for execution. We acknowledge that this limits any potential use of our emulator for educational purposes, but it consequently focuses its abilities on large-scale research-oriented emulation. Thus more attention is given to such application, by providing additional routines for non-interactive actions on the containers (asynchronous commands, file download from and upload to the containers, etc.)

In addition, connection to the containers in Bignet is not achieved by a management network. Instead, all sent commands and all received results and files go through the host machines which act as gateways. Specifically, Bignet only maintains SSH sessions with the physical hosts, which are in charge of forwarding instructions to the containers via `docker exec` commands (Figure 1). This is an important key point and is essentially what makes Bignet much more scalable than all the existing counterparts.

Bignet does not innovate in the emulated network mapping aspect of distributed emulation. Instead, it implements basic algorithms (random and round-robin mapping) as well as the optimised heuristics developed by Distrinet’s authors. It also offers an interface for implementing new mapping algorithms, potentially ones custom-made for the desired emulated scenario.

Building on the comparison detailed in [4] between Distrinet, Mininet Cluster Edition, and Maxinet based on pre-defined criteria, table I summarises how Bignet compares to these distributed emulators with regards to the design principles discussed above (compatibility with Mininet’s API, and virtualisation/containerisation technologies).

b) Performance Evaluation: In this section we present a series of performance evaluation tests to compare key performance indicators between Bignet and Distrinet. In particular, we show how the minimalist design and implementation of

	Mininet CE	Maxinet	Distrinet	Bignet
Mininet API				
Python scripts	Yes	Yes	Yes	Yes
Interactive shell	Yes	Yes	Yes	No
Virtualisation				
End-hosts	Linux ns	Linux ns Docker	LXC	Docker
Switches	OvS	OvS	OvS in LXC	OvS

TABLE I: Comparison of different distributed network emulators along the design principles.

Bignet –particularly with regards to how virtual switches are not containerised and how the links are emulated using fewer virtual links (Linux virtual ethernet pairs)– allows it to sustain larger emulated networks with more traffic and less delay.

To compare traffic emulation performance between Bignet and Distrinet, we conduct the following experiment on a single host¹: a server sends a heavy TCP flow to a client through a variable-length network of cascading switches. All links are emulated with neither limits on bandwidth nor simulated delay, essentially to show the maximum speed and size of traffic that the emulated switches can forward. For each number of intermediate switches, we send a 100 seconds-long TCP flow and monitor a) the average throughput defined as the total volume of traffic sent and received, and b) the smallest recorded delay over all packets in the exchange.

Figure 2 shows the results. We can see how Bignet achieves (on average) more than Distrinet in terms of maximum throughput, and less in terms of delay. We also observe that the gains in performance increase and the supremacy of Bignet gets more statistically significant as we add more switches, up to 1 Gbps more throughput and 15 microseconds less delay for a network of 10 intermediate switches. These gains in performance are mainly due to Bignet’s shorter emulated data paths as the virtual switches run directly on the physical machines and do not need to be isolated inside their own containers.

The most important goal behind the development of this new lightweight distributed network emulator is to overcome the observed scale limits of Distrinet. To show how this has indeed been achieved by our shift from Distrinet’s original design principles and implementation technologies, we perform another experiment: we emulate a variable-length linear topology² on a cluster of 10 machines using a round-robin mapping algorithm³. In this experiment, the key performance indicator is the start-up time, defined as the time required for all the nodes of the emulated network to start running, and which is considered infinite if it exceeds one hour or if the emulator fails to set-up the emulation and halts. For each

¹All experiments were conducted in the UVB cluster of Grid5000’s Sophia site. More information about its hardware can be found at <https://www.grid5000.fr/w/Sophia:Hardware>.

²Mininet’s famous linear topology is a cascade of n switches where one end-host is connected to each switch. A linear topology with parameter n therefore totals n switches, n end-hosts, and $2n - 1$ links.

³The round-robin mapping algorithm distributes the emulated end-hosts and switches by cycling through the set of available hosting machines. In other terms, in an infrastructure of N machines, switch i and end-host i will be mapped to physical host $i \bmod N$.

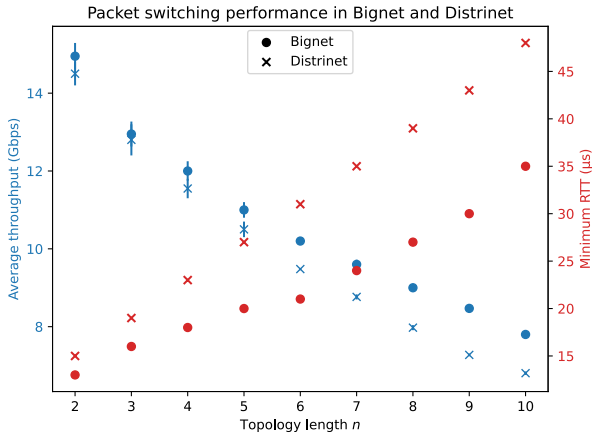


Fig. 2: Maximum achievable throughput and minimum possible delay (y-axis) in a topology of many cascading switches (x-axis) emulated using Bignet and Distrinet.

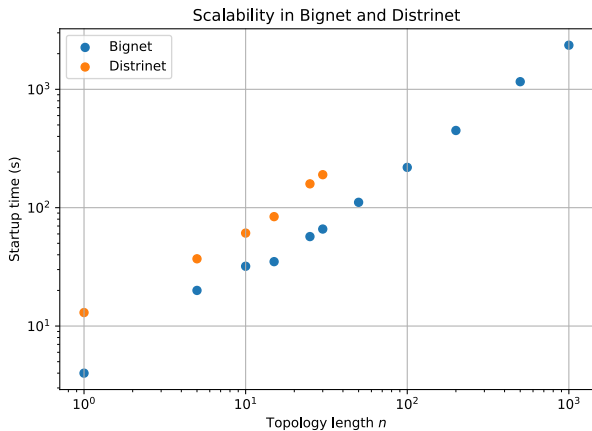


Fig. 3: Startup time of variable-length linear topologies emulated using Bignet and Distrinet.

value of n we repeat the experiment 10 times and compare the smallest out of 10 recorded start-up times.

Figure 3 shows the results. Indeed, Bignet proves to be faster in setting-up the emulated networks for all sizes, and can set up networks of more than 2000 nodes (1000 end-hosts and 1000 switches), while Distrinet cannot emulate more than 60 nodes ($n = 30$).

3. Conclusion

In this section we presented Bignet: a lightweight distributed emulator inspired from Mininet (and particularly its distributed version Distrinet). Bignet is the result of a shift from the current paradigm, both its design principles and its technological choices, and which trades off self-imposed constraints (compatibility with Mininet, use of Linux-native tools) for concrete increases in scalability and performance. We have shown that Bignet performs better than Distrinet –a very powerful Mininet-like distributed emulator– in terms of traffic speed and deployment scale.

Bignet is not aimed at replacing the currently available emulators but only serves as a proof-of-concept to demonstrate how they could be improved to achieve their original goals: performance and scalability. In the next sections, we will discuss the problem of emulation accuracy and will propose a framework for fidelity monitoring of network emulation. This will culminate in Hifinet (High-fidelity networks), an enhanced version of Bignet that implements the designed fidelity framework.

III. HIGH-FIDELITY NETWORK EMULATION

In this section, we will present a theoretical framework for assessing the fidelity of network emulations, both single-machine and distributed. The end goal of this first step is only to mitigate the inherent limits of realism by informing the user whether emulation failures during the experiment may have negatively impacted the results and, subsequently, whether the results should be discarded. We will first provide a conceptual and operational definition of emulation fidelity, then derive an axiomatic approach to fidelity monitoring through passive delay measurement, and finally conclude with a proof-of-concept implementation to show how it performs in a concrete scenario.

1. Emulation Fidelity

a) Definition: Intuitively, fidelity (or realism) in the context of network simulation and emulation is a measure of the quality of modeling and replication of networks and their components. However, it is a challenge to sketch a rigorous definition that is both *conceptual* –giving an abstract frame to reason about the concept of fidelity and its relationships to other concepts, and *operational* –allowing practical use in measurement and monitoring.

In this paper, we distinguish between two types of definitions of fidelity:

- those that attempt a *noumenal* definition, characterising the modeling aspect of emulation and measuring how close the internal behaviour of the emulated network is compared to the target production environments; and
- those that attempt a *phenomenal* definition, instead considering and comparing only certain high-level aspects of the real and emulated networks, which are perceived as phenomena by the user and which can be modeled and measured to assess similarity.

For instance, it is possible to compare an emulated router to the hardware router it emulates based on whether or not it runs the same software and operating system, whether it is allocated the same number of CPU cores, whether or not it implements the same packet processing technologies, etc.: this is noumenal fidelity. On the other hand, the emulation can also be assessed based on how much traffic both routers process, how much delay do they incur, etc.: this is phenomenal fidelity. Essentially, phenomenal fidelity is a weaker aspect of fidelity that only focuses on what the user can perceive and measure, and does not concern itself with low-level, internal behaviour.

The following subsections propose and explore definitions of the latter type. These have an important perk of being both

conceptual and operational by design, as they can be measured and monitored exactly through the set of phenomena on which they are defined.

b) Phenomenal Assessment of Emulation Fidelity:

Knowing that network emulation is defined by the simulation of the hardware and of the communication media, it is sufficient to only consider these components for the evaluation of fidelity. We can thus choose network phenomena that directly translate their behaviour. These network phenomena are particularly convenient due to being *universal*, as they do not depend on the experimented scenario, and do not particularly raise challenges in their implementation. They are the *link bandwidth*, the *packet loss*, the *packet ordering*, and the *packet delay*.

To better define fidelity with regards to these phenomena, we first consider a phenomenal model of a network, defined as a set of unidirectional links (the communication media). Each link is defined by:

- Two ends (1) and (2) respectively for transmission and reception;
- A bandwidth B , a propagation delay π , and a loss rate λ ;
- Three finite sets of packets \mathcal{P}_0 , \mathcal{P}_1 , and \mathcal{P}_2 , containing the generated (from the upper layers), transmitted, and received packets respectively, and such that $\mathcal{P}_2 \subseteq \mathcal{P}_1 \subseteq \mathcal{P}_0$, additionally equipped with a size operator $|\cdot|$, and such that:

$$\frac{|\mathcal{P}_0| - |\mathcal{P}_1|}{|\mathcal{P}_0|} = \lambda; \quad (1)$$

- Two clock functions $t_1 : \mathcal{P}_1 \rightarrow \mathbb{R}_+$ and $t_2 : \mathcal{P}_1 \rightarrow \mathbb{R}_+ \cup \{\infty\}$, giving the timestamps of enqueueing of packets at the transmission and receiving at the reception, with $t_2(P) = \infty$ for all packets $P \notin \mathcal{P}_2$;
- A function $\delta : \mathcal{P}_1 \rightarrow \mathbb{R}_+$ associating to each packet P its delay $\delta(P)$, and which must satisfy the recursive formula for any two successively transmitted packets $P_i, P_{i+1} \in \mathcal{P}_1$:

$$\delta(P_{i+1}) = \pi + \frac{|P_{i+1}|}{B} \quad (2)$$

$$+ \max [0, \delta(P_i) - \pi - (t_1(P_{i+1}) - t_1(P_i))]. \quad (3)$$

The last equation is essentially a recursive formulation of the canonical delay model in wired networks. It states that the delay of a packet P_{i+1} is the sum of its propagation delay π along the link and its transmission delay by the hardware if it is not to wait in the queue; and otherwise is equal to the sum of its transmission delay, the total delay of the previous packet P_i excluding the duration of time that it has spent in the queue before P_{i+1} 's arrival.

Link bandwidth is one of the most basic link-level network phenomena to monitor in order to assess the accuracy in emulating a wired communication medium, and by extension the fidelity of network emulation. Without knowledge about the rate at which packets are generated and prepared for transmission by the link, it is not possible to verify that the link capacity is fully utilised at all times. However, it is possible and easy to check that the capacity limits the rate

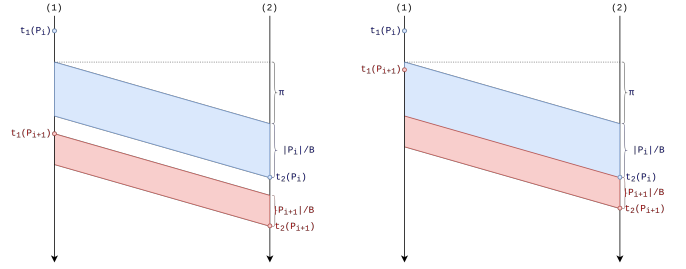


Fig. 4: Queuing, transmission, and reception of two successively sent packets P_i and P_{i+1} in two cases: packet P_{i+1} is not queued (left); and P_{i+1} waits in the queue before transmission (right).

at which packets are transmitted. In our phenomenal model of the network link, this can be formulated as follows:

Definition 1. An emulated link is said to be **bandwidth-accurate** if, for any two successive packets $P_i, P_{i+1} \in \mathcal{P}_1$,

$$t_2(P_{i+1}) - t_2(P_i) \geq \frac{|P_{i+1}|}{B},$$

(with equality when packet P_{i+1} enters the system before P_i is fully transmitted),

which essentially ensures that no two packets are received faster or slower than the bandwidth permits. The examples in Figure 4 show the case of equality (right) and strict inequality (left).

Packet loss is another basic link-level network phenomenon that can be monitored for link accuracy assessment. One may simply count the ratio of lost packets (generated but not received) and compare it to the loss rate of the emulated link. In our model, this can be formulated as:

Definition 2. An emulated link is said to be **loss-accurate** if

$$\frac{|\mathcal{P}_0| - |\mathcal{P}_2|}{|\mathcal{P}_0|} = \lambda.$$

Note that such formulation is ideal as in practice the ratio can, at best, only approach the configured loss rate as the size of \mathcal{P}_1 approaches ∞ . Another important caveat is that only losses due to the unreliability of the communication medium should be accounted. In other words, any loss due to congestion should be recognised separately.

Our phenomenal model of the link also allows packet re-ordering to be a metric through which link emulation accuracy can be assessed:

Definition 3. An emulated link is said to be **order-accurate** if, for any two packets $P_i, P_j \in \mathcal{P}_1$,

$$t_1(P_i) < t_1(P_j) \implies t_2(P_i) < t_2(P_j),$$

which ensures that packets are always received in the order in which they are transmitted.

The last metric our phenomenal model can define is delay accuracy. Its definition within our model is straightforward:

Definition 4. An emulated link is said to be **delay-accurate** if, for any packet $P \in \mathcal{P}_1$:

$$t_2(P) - t_1(P) = \delta(P).$$

c) *Delay-based Fidelity Monitoring*: The previous definitions of link emulation accuracy can be seen as different phenomenal aspects of emulation fidelity of communication media. The four metrics can be monitored separately or jointly to provide evidence of good phenomenal fidelity, but they are in fact loosely correlated. Indeed, we will prove in this section that the delay is the finest of the four phenomena, in the sense that it captures information contained in the others, and develop our fidelity monitoring framework around it.

Definition 5. A phenomenon X is said to be **finer** than a phenomenon Y if X -accuracy necessarily ensures Y -accuracy.

- Properties 1.**
- 1) Delay is finer than bandwidth.
 - 2) Delay is finer than order.
 - 3) Delay is finer than loss.

Proof. 1) Suppose that an emulated link is delay-accurate. Let P_i and P_{i+1} be two successively transmitted packets. From 2, we have:

$$\delta(P_{i+1}) \geq \pi + \frac{|P_{i+1}|}{B} + \delta(P_i) - \pi - (t_1(P_{i+1}) - t_1(P_i)),$$

which simplifies to:

$$\delta(P_{i+1}) - \delta(P_i) + (t_1(P_{i+1}) - t_1(P_i)) \geq \frac{|P_{i+1}|}{B}.$$

Since the link is delay-accurate, we also have:

$$t_2(P_{i+1}) - t_2(P_i) = \delta(P_{i+1}) + t_1(P_{i+1}) - \delta(P_i) - t_1(P_i),$$

and therefore

$$t_2(P_{i+1}) - t_2(P_i) \geq \frac{|P_{i+1}|}{B}$$

which concludes the proof. Consequently, when P_{i+1} arrives before P_i is fully transmitted, the equality holds and we have:

$$\delta(P_{i+1}) = \pi + \frac{|P_{i+1}|}{B} + \delta(P_i) - \pi - (t_1(P_{i+1}) - t_1(P_i)),$$

and therefore, by following the same calculus,

$$t_2(P_{i+1}) - t_2(P_i) = \frac{|P_{i+1}|}{B}.$$

- 2) To prove that delay is finer than order, it is sufficient to prove that the order is not broken for pairs of successive packets. Suppose that a link is delay-accurate and let P_i and P_{i+1} be such a pair. From property (1), we know that the link is also bandwidth-accurate, and thus

$$t_2(P_{i+1}) - t_2(P_i) \geq \frac{|P_{i+1}|}{B} > 0,$$

which concludes the proof.

- 3) Suppose that a link is delay-accurate. Loss-accuracy is equivalent to proving that all losses occur between generation and transmission and that no additional packet is lost between the transmission and reception, i.e. $\mathcal{P}_1 = \mathcal{P}_2$. Let P be a packet in \mathcal{P}_1 . Since the link is delay-accurate,

$$t_2(P) = t_1(P) + \delta(P) < \infty,$$

and thus $t_2(P)$ is well defined in \mathbb{R}_+ which, from our model, supposes that $P \in \mathcal{P}_2$ and concludes the proof. \square

These proofs show that all the considered network phenomena can be brought down to delay, which triumphs as the finest metric for link emulation accuracy –and thus network emulation fidelity– evaluation. Although this has only been formally proven within the frame of our approximate theoretical model, it demonstrates a broader truth: that all link-level network phenomena are ultimately features of time and can thus be formalised in temporal language. Packet reordering can be understood as an inconsistency in the timestamps recorded by the two ends of the link; packet loss as infinite delay; and even bandwidth can be expressed in temporal terms. Other than the four listed above, many metrics that are important to network specialists may also be directly or indirectly tied to the delay. The jitter, for instance, is a direct feature of delay and is therefore accurately emulated as long as the delay is too. Indeed, if the delay of each packet individually is accurate, then all its statistical moments, including the jitter, for any sample of packets, are also accurate. Knowing this, the delay imposes itself as the superior criterion for phenomenal evaluation of fidelity.

The corollary of all the above reasoning is that delay is a good network metric on which we can build a definition of phenomenal fidelity. In principle, it is both conceptually and operationally valid. But in practice, however, it is difficult, and arguably impossible, to judge fidelity based on the ideal equation between timestamp difference and modeled delay. Neither emulators nor systems of measurement can achieve such perfection. Thus we first redefine delay-accuracy to allow some margins of inevitable error:

Definition 6. Let $\gamma, \epsilon \in (0, 1)$. An emulated link is said to be (γ, ϵ) -**delay-accurate**, or **probably approximately delay-accurate with parameters** (γ, ϵ) if, for any packet $P \in \mathcal{P}_1$ and with probability at least $1 - \gamma$,

$$|(t_2(P) - t_1(P)) - \delta(P)| \leq \epsilon.$$

In higher-level terms, the principle can be formulated as the following criterion:

Criterion 1. For an emulation to have good fidelity, the deviation of the measured delays from the model delays of a sample set of packets throughout the duration of the experiment should not be too large.

The criterion for fidelity raises important caveats regarding its implementation that we discuss individually in the remainder of this section.

d) *Challenges*: The *model delay* used as baseline against which we compare the measured delays raises an important caveat regarding its estimation. One way to estimate this theoretical/expected delay is to use the standard delay model canonically established in the literature:

$$\delta(P) = \frac{l}{v} + \frac{|P|}{B} + \frac{|Q(P)|}{B},$$

where the first term is the propagation delay (independent of the packet) configured by the user, the second term is the

transmission delay which depends on the size of the packet and the configured bandwidth (or transmission speed), and the last term is the queuing delay which depends on the size of the queue when the packet enters and the configured bandwidth. All of these parameters can be known to the fidelity monitoring tool, and logged alongside the timestamps of arrival, departure, and reception of the packet. This equation perfectly models the behaviour of wired links given certain known variables, but needs to be carefully adapted to radio channels and other wireless media.

Another issue is that of passive delay measurement. This has been tackled in [5] where we have argued that one-way delay can only be passively measured under strict time synchronisation assumptions, and where we have proposed an alternative methodology to measuring the round-trip delay instead. Because no assumption about time synchronization is reasonable in our context, we will stick to measuring the round-trip delays (RTD) of pairs of packets P_1, P_2 which is to be compared against the sum of their own delays $\delta(P_1) + \delta(P_2)$ expected from a correct emulation. The major downside of such a choice is that information about the individual one-way delays (OWD) is lost in the RTD. As such, an assessment made from a certain measurement is made on both the packets of the pair.

Regarding overhead at scale, measuring the delays of all packets over all emulated links in the network is neither possible nor necessary. While it is not intrusive to the emulation itself even at extreme scales (see the next subsection), it can generate absurd amounts of data which can be hard to store, transfer, and analyse. It is therefore sufficient to randomly sample a subset of packets, or implement an intelligent sampling strategy that focuses more on low-delay packets (small size and/or small queue length) or high-bandwidth links, where errors are more likely to happen and be detected. In our implementation, we use random hash-based sampling [12] applied to packet headers to make sure that a packet's information is logged at both ends of a certain link, even if they are hosted on different machines.

Another important caveat is that the deviation between measured and model packet delays can be evaluated using different statistical metrics. A straightforward approach is to consider the mean absolute error (MAE) as a measure of deviation between measured values ($\overline{RTD}(P_1, P_2)$) and expected values ($RTD(P_1, P_2) = \delta(P_1) + \delta(P_2)$). The emulation can then be considered incorrect if the MAE (over all considered pairs of packets) exceeds a certain threshold established beforehand and which expresses how much fidelity is expected from the emulation. If the user is not able to decide on such a threshold, then they can instead consider the mean *percentage* absolute error (MPAE) by measuring the deviation relative to the model values. The user can then work with a *universal* threshold value (such as 1% or 5% for strict fidelity standards, or up to 50% for looser ones).

However, these two metrics share the common drawback of being measures of *averages* and do not consider values individually, which leads to higher errors being compensated by lower ones. Thus, in the presented version of our framework, we consider quantiles: the emulation will be presumed

correct if a certain number of measured values (e.g., 95% of all measures) do not deviate from the estimations by more than the threshold error value. Here again it is possible to reason in terms of relative error⁴, but while this is certainly an advance compared to averages, it still treats all packets with equal importance and makes an assessment based on the overall measure of quantile. This is efficient and more precise but not yet ideal as major errors happening in a short period of time and on a limited number of packets can have propagating macro-level effects on the emulation while being considered negligible by the system at the scale of the entire set of packets. In this case the user can consider looking at the *sliding time window quantiles*, i.e., by assessing the experiment to be accurate if, given a number K (resp. time window T), the condition holds for all sequences of K successive pairs of sampled packets (resp. for all pairs of sampled packets in any time window T).

2. Implementation

Our solution for the measurement and estimation of delay is built up from three main components.

a) Packet loggers: This component is a collection of eBPF functions that are plugged into a number of specific kernel routines and which therefore run in kernel space. Its goal is to capture and log information about sampled packets in persistent storage (Figures 5 and 6). After a message is made into a packet and then into a Linux data structure, it is enqueued by the TC subsystem –provided the queue is not full– and waits for a period of time before being dequeued and sent to the virtual NIC for transmission, or randomly dropped with a certain probability to simulate loss if it is enabled. And if the packet is to be successfully transmitted, the packet logger logs in raw files information about its enqueue event (event timestamp, packet size, and the length of the queue at the packet's arrival) and dequeue event (event timestamp) if it is sampled for monitoring.

Specifically, using eBPF we embed low-level instructions into the TC datapath, at both ends of each virtual link, which run whenever a packet is received by the TC subsystem. This ensures that our passive packet monitoring methodology incurs no significant computing overhead on the kernel (particularly networking) and on application processes.

b) Local monitoring agents: These agents are intermediary user space programs that run on the hosts and whose goal is to parse the logs from the packet loggers and compile them into tables that can later be used for analysis. This is executed after the emulated experiment has finished running, and therefore does not interfere with the emulation.

c) A collector/analyser: This component is the brain of the system. Its job is to collect and analyse packet information compiled by the monitoring agents. It is logically unique and achieves its goal in two steps:

⁴The quantile percentage absolute error (QPAE) is thus a network-level transposition of the *probably approximately delay-accurate* concept we have previously defined for single emulated links: the quantile being the probability parameter γ and the threshold being the approximation parameter ϵ .

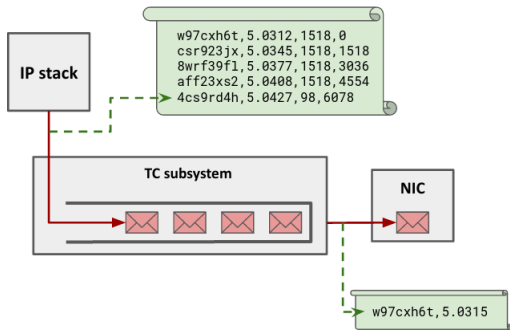


Fig. 5: TC datapath interception by packet loggers.

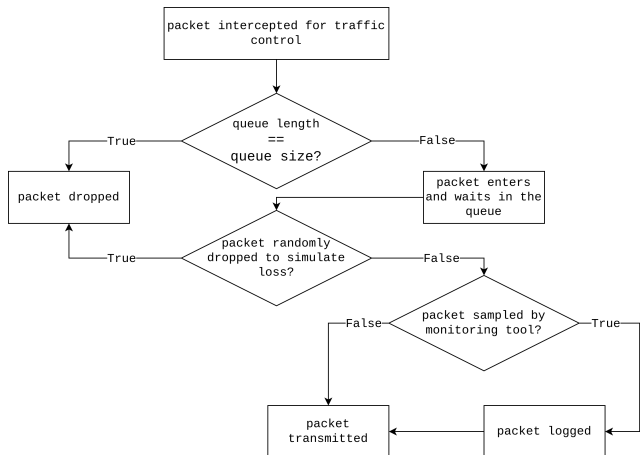


Fig. 6: Interception and logging of packets.

- First, the data is collected from the monitoring agents as tables, which are then cross-examined to match information about packets distributed over multiple tables. For instance, a table from one monitoring agent (and therefore from one machine of the cluster in the case of distributed emulation) can contain the sending timestamp of a packet, and another table from a different monitoring agent can contain the reception timestamp of the same packet. The output of this step is a unique large table where each entry corresponds to a packet and is identified by its ID, and which contains all information about it;
- Finally, packets from the table are paired together according to a pairing rule and their joined RTD is measured (from the logged timestamps) and estimated (from other information such as packet sizes, queue lengths, etc.). These two values are then compared for all considered pairs of packets and an overall judgement about the emulation can be made.

3. Overhead

In the previous subsection, we have demonstrated the implementation of our methodology, which is designed in a distributed and hierarchical fashion: a central and logically unique component analyses the monitoring data sent from multiple local agents, which in turn gather their data from lightweight packet loggers. In practice, the central collector/analyser and the local monitoring agents perform their tasks –of analysis

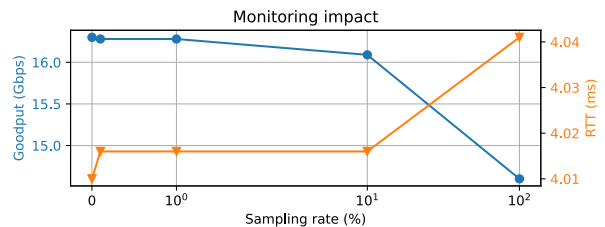


Fig. 7: Impact of monitoring on the emulation performance. The orange plot shows the average achieved goodput and its small variance and the blue plot shows the minimum RTT.

and data collection– *offline*, i.e., after the emulation has completed, and therefore do not disturb its execution. However, as packet loggers intercept the transmission of emulated packets on which they perform certain processing instructions, these can cause overhead by reducing the switching capacity of the emulated network (defined as the amount of packets that can be transmitted per unit of time) and/or inflict additional delay in the emulated links. We have argued that a sampling strategy can trade off a narrow decrease in statistical accuracy of the results to mitigate these overheads, which we show experimentally in this subsection.

To do this, let us consider the following emulated network: an emulated switch connects an emulated server to an emulated client with unlimited capacity (i.e., no traffic control is used to limit the bandwidth) and a 1 ms propagation delay. In a first run, we send a heavy long Iperf TCP flow from the server to the client and record the average achieved goodput over a window of 100 seconds. In a second run, we send 10 000 Ping echoes initiated by the client and record the minimum RTT. These experiments are emulated on a single physical host and use our fidelity monitoring tool with random packet sampling, and repeated for different sampling rates: 100%, 10%, 1%, and 0.1%. The former metrics (average Iperf goodput and minimum Ping RTT) are compared across the different sampling rates, and against a setting where the monitoring tool is turned off (corresponding in the following figure to a sampling rate of 0%).

The results of this evaluation are shown in Figure 7. As expected, the throughput performance of the emulation decreases when the tool is used (from 16.30 Gbps down to 14.60 Gbps for a sampling rate of 100%), while the delay per packet slightly increases (few microseconds of overhead). These changes depend on the packet sampling rate configured for the packet loggers, and seem to follow the performance-sampling law found in [7]. Overall, the observed relatively small drops in performance prove that our fidelity monitoring implementation does not paralyse the emulation even at large sampling rates. In all following experiments, we will configure a sampling rate of 10% which does not impact networking capacities by more than 1%.

4. Evaluation

So far we have presented the design of our delay-based monitoring framework and shown evidence of its small impact on performance. The underlying principle is that emulation

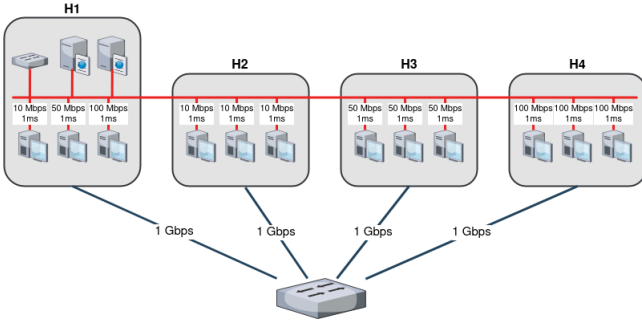


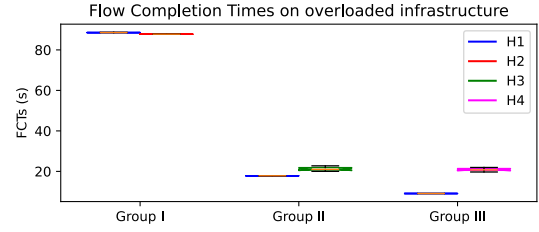
Fig. 8: Emulated network (red) and underlying cluster network. Clients from Group I are emulated in $H1$ and $H2$; from Group II in $H1$ and $H3$; and from Group III in $H1$ and $H4$.

failures manifest as inaccurate emulation of delay that leads to higher-level errors which can compromise the overall results of the experiment. In this section we show through an example how this assumption performs in practice. More specifically, we present a common network emulation scenario and correlate the delay monitoring metrics with application-level metrics.

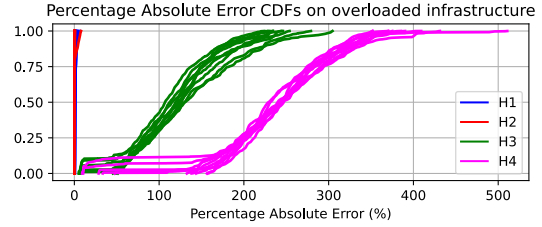
a) Testbed: In this scenario, a set of $N = 3n$ clients are synchronously downloading a 100 MB file from a random server (out of 5) located on the same Ethernet segment. The client hosts are separated into three groups: n clients from Group I are connected to the switch by 10 Mbps-bandwidth and 1 ms-delay links; n clients from Group II by 50 Mbps-bandwidth and 1 ms-delay links; and n clients from Group III by 100 Mbps-bandwidth and 1 ms-delay links. The servers are connected by links with no traffic control. The experiment is run using the latest version of DistroNet to date (v1.2) on four nodes of the R2Lab cluster⁵, which are connected in a star topology to one single switch (Figure 8). Furthermore, our embedding algorithm is configured in a way that all emulated file servers and the virtual switch are hosted in the same machine (host $H1$); the emulated clients from Group I are hosted in $H1$ and $H2$, from Group II in $H1$ and $H3$, and from Group III in $H1$ and $H4$ (see Figure 8).

The idea is to compare the flow completion times (FCTs) and the measured delay errors between and within the groups. If this scenario were emulated with perfect fidelity (i.e., behaving exactly as it would in real networks), (a) there should be no difference in the FCTs within each group as all clients from the same group are equivalent in the emulated topology, regardless of whether or not they are hosted *locally* with the server, and (b) clients in Group I (10 Mbps bandwidth links) should experience the largest FCTs, followed by clients in Group II (50 Mbps bandwidth links) and finally the clients in Group III (100 Mbps bandwidth links) should experience the lowest FCTs.

b) Run 1: Overloaded Infrastructure: In this first run of the scenario, we emulate a total of $N = 40$ clients distributed over the 4 hosts. Figures 9 show the results. The first thing to note is how the clients in Groups II and III experience different FCTs depending on whether they are hosted in $H1$



(a) Flow Completion Times for the clients of each group. From left to right: Group I, Group II, and Group III. The left-side box plot shows the FCTs of clients hosted in host $H1$ (blue), and the right-side box plot for clients in hosts $H2$ (red), $H3$ (green), and $H4$ (magenta). Each box plot shows all 4 quartiles.



(b) CDFs of the percentage absolute errors (PAE). The blue plots correspond to the CDFs of locally emulated links in host $H1$; the red, green, and magenta to the CDFs of links overlay emulated between hosts $H2$ and $H1$, hosts $H3$ and $H1$, and hosts $H4$ and $H1$ respectively.

Fig. 9: High-level (a) and low-level (b) indicators of emulation fidelity.

or $H3$ and $H4$ (figure 9a). In particular, these get an average of 17.81 seconds vs 21.24 seconds for Group II, and 9.08 seconds vs 20.92 seconds for Group III. This is also evident from the CDFs of percentage absolute error: overlay links for clients in hosts $H3$ and $H4$ experience higher relative delay emulation error (figure 9b).

This example essentially demonstrates how higher-level incorrect behaviour due to underlay congestion, which occurs silently and which can lead to false analyses, is in fact correlated with *objectively* incorrect lower-level behaviour easily perceivable from a delay perspective. In this example, the differences in delay emulation errors between local and overlay links are mainly due to the additional delay that emulated packets experience as they cross the infrastructure network. To troubleshoot the causes behind this perceived inaccuracy, it is important to see that the bandwidths of all overlay links sum to a total of 1.6 Gbps, while all these emulated links have to cross the physical link connecting the cluster switch to host $H1$, whose capacity is limited to 1 Gbps. The congestion control algorithm (CUBIC TCP) used by clients to download the file distributes the available bandwidth in a way that the throughput of greedy clients (Group II and III hosted in $H1$) is lowered: clients from Group I get a throughput of around 10 Mbps (equal to their bandwidth), clients from Group II get a throughput of around 45 Mbps (90% of their bandwidths), and clients from Group III get a throughput of around 45 Mbps (45% of their bandwidths). This results in clients from Groups II and III hosted in $H3$ and $H4$ getting longer FCTs than their counterparts hosted in $H1$.

⁵Reproducible Research Lab: <https://r2lab.inria.fr/index.md>.

And while clients from Groups II and III hosted in $H3$ and $H4$ get the same throughput (and thus experience the same FCTs), the links connecting them to the Ethernet switch show different PAEs: the median for links emulated between $H3$ and $H1$ is around 119%; while the median for links between $H4$ and $H1$ is around 238%. This is due to the former links having a higher bandwidth –and thus their packets a lower RTD on average– while both experiencing approximately the same added delay, which leads to different errors in relative values.

c) Run 2: Impact of infrastructure load: Emulating this scenario in such an infrastructure might seem artificial, and while it does demonstrate the failures of distributed network emulators in certain settings and how those failures can be captured by our delay monitoring framework, such settings might appear unrealistic at first glance: to avoid these problems the user need only analyse the capacities of the infrastructure and distribute the emulated nodes accordingly. However, this is not always possible as the user may be using a shared infrastructure –cloud or grid– over which they have a very limited amount of control and/or knowledge. In such cases, the maximum bandwidth of each physical link may be disclosed, but the fraction available to the user at all times is generally not.

Nevertheless, the delay emulation error highly correlates with higher-level inaccuracies independently of the infrastructure usage. In this scenario, the load⁶ ρ on the physical link connecting host H_1 to the switch S can reach 160% ($N = 40$):

$$\rho = \frac{(10 + 50 + 100) \cdot \frac{N}{4}}{1000} = 160\%.$$

In this second run, by varying the number of emulated clients N , we can vary this maximum load, and observe different degrees of high-level and low-level emulation infidelity for values below or above 100%. Figure 10 shows how these two indicators correlate for different loads (their Pearson correlation coefficient is approximately equal to 0.91). The *deviation* is a chosen application-level metric that measures the relative difference in FCTs between clients from group III hosted in the same machine (H_1) as the server, and clients from group III hosted in H_4 . Also note that delay errors increase much faster when the emulated network approaches the underlay capacities, signaling early that failure should be expected.

IV. CONCLUSION

This paper has tackled the complex problem of scientific experimentation in the context of distributed systems and computer networks, specifically by addressing network emulation as one of its paradigms. The initial hypothesis was that emulation is the best approach, especially using lightweight container-based network emulators such as Mininet. Indeed, such software makes the setup and operation of complex networks and intricate scenarios easy to accomplish and easy to share. We raised however the issues of scalability and realism:

⁶The load or the usage of a link is defined here as the volume of traffic it transports relative to its bandwidth.

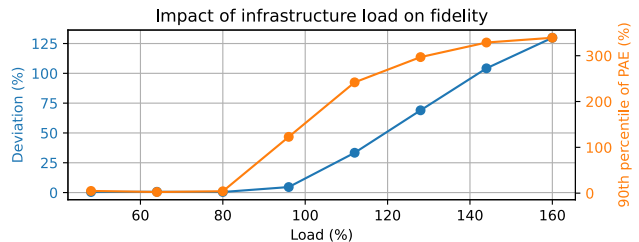


Fig. 10: High-level (blue) and low-level (orange) indicators of emulation fidelity (y-axis) vs. link load (x-axis).

as all the components of the emulated network share the same pool of resources running a large scenario is a delicate task; and as the emulator relies on software tools –based on fallible models– to simulate some pieces of the network the user cannot expect perfect fidelity of behaviour. We addressed the former by presenting a new distributed emulator destined for extreme-scale network scenarios, and the latter by proposing the concept of network fidelity and developing a framework for its monitoring and troubleshooting. We demonstrated that our proposed tools achieve their objectives to the best extent and perform better than what is existent.

Though our tools and methodologies were powered by the wider literature on network measurement and modeling, the presented work attempts to deal with a novel concept (fidelity) in a very niche area (network emulation). We hope this will inspire further research into the topic, which we believe can tremendously help network researchers produce better quality scholarships. In particular, we believe the following two axes to be interesting initial questions for such endeavour:

- We have theorised the concept of emulation fidelity but only focused on its measurement at the phenomenal level. We believe that tools and theories from formal software and hardware specification and verification (e.g., propositional calculus, automata theory, and program semantics) can be used to monitor stronger constraints on fidelity. Such models would be somewhat close to our concept of noumenal fidelity, and could inform with more certainty whether emulation results are more accurate. The only drawback would be scalability, where all the challenge in researching this subject lies;
- We have presented in this paper a framework for emulation fidelity monitoring that only informs the user whether failures have happened that may potentially compromise emulation results. The logical next step is to attempt a troubleshooting of the underlying causes in order to avoid them and reconduct the emulated experiment under better circumstances. To achieve this, we have explored an approach inspired from delay tomography where we attempt to extract infrastructure load through infrastructure delay information extracted from the emulated delay measurements. The approach transforms the emulation troubleshooting problem into a mathematical programme whose solutions inform about the state of the individual components of the underlying infrastructure.

REFERENCES

- [1] Mininet: an instant virtual network on your laptop (or other pc), 2012.
- [2] Mininet cluster edition, 2016.
- [3] Carl Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.
- [4] Giuseppe Di Lena, Andrea Tomassilli, Damien Saucez, Frédéric Giroire, Thierry Turletti, and Chidung Lac. Distrinet: A mininet implementation for the cloud. *ACM SIGCOMM Computer Communication Review*, 51(1):2–9, 2021.
- [5] Houssam Elbouanani, Chadi Barakat, Walid Dabbous, and Thierry Turletti. Passive delay measurement for fidelity monitoring of distributed network emulation. *Computer Communications*, 195:40–48, 2022.
- [6] Houssam Elbouanani, Chadi Barakat, Walid Dabbous, and Thierry Turletti. Delay-based fidelity monitoring of distributed network emulation. In *Proceedings of the TASIR Workshop: Testbeds for Advanced Systems Implementation and Research*, 2023.
- [7] Houssam ElBouanani, Chadi Barakat, Guillaume Urvoy-Keller, and Dino Lopez-Pacheco. Collaborative traffic measurement in virtualized data center networks. In *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, pages 1–3. IEEE, 2019.
- [8] Brandon Heller. *Reproducible network research with high-fidelity emulation*. Stanford University, 2013.
- [9] David Muelas, Javier Ramos, and Jorge E Lopez de Vergara. Assessing the limits of mininet-based environments for network experimentation. *IEEE Network*, 32(6):168–176, 2018.
- [10] Javier Ortiz, Jorge Londoño, and Francisco Novillo. Evaluation of performance and scalability of mininet in scenarios with large data centers. In *2016 IEEE Ecuador Technical Chapters Meeting (ETCM)*, pages 1–6. IEEE, 2016.
- [11] Philip Wette, Martin Dräxler, Arne Schwabe, Felix Wallaschek, Mohammad Hassan Zahraee, and Holger Karl. Maxinet: Distributed emulation of software-defined networks. In *2014 IFIP Networking Conference*, pages 1–9. IEEE, 2014.
- [12] Tanja Zseby, Maurizio Molina, Nick Duffield, Saverio Niccolini, and Fredric Raspall. Sampling and filtering techniques for ip packet selection. Technical report, 2009.