Delay-based Fidelity Monitoring of Distributed Network Emulation

Houssam ElBouanani, Chadi Barakat, Walid Dabbous, and Thierry Turletti
Inria, Université Côte d'Azur

2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS) | 978-1-6654-7706-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/COMSNETS56262.2023.10041407

***Abstract*—Distributed Network emulators (e.g., Mininet Cluster Edition) have proven to be an attractive solution to perform extreme-scale network and systems evaluation on smaller-size testbeds and experiment platforms. They can provide contained, customisable, and scalable testing environments for researchers to evaluate their contributions and reproduce their results. The major drawback of this approach in network experimentation is the use of virtual components (hosts, network switches, etc.) that do not behave with perfect similarity to the physical components they emulate, mainly due to the concurrency in using the underlay network and computing resources. We thus present in this paper a methodology to monitor emulation fidelity by measuring the network delays of emulated packets, which relies on statistical metrics to evaluate their inaccuracy. We further dig into the possible sources of emulation inaccuracy and show how our system can detect them to avoid biased experiment results. We particularly show through a common experiment scenario how undetected network emulation errors can lead to biased results.**

***Index Terms*—network emulation, passive monitoring, delay measurement**

I. Introduction

Distributed network emulation is a new paradigm for running experiments for computer networks and distributed systems research. This paradigm complements traditional testbeds and experiment platforms with versatility and scalability. Indeed, a distributed network emulator (e.g., Mininet Cluster Edition [1], Maxinet [13], and Distrinet [5]) allows users to run reproducible, complex, and large-scale experiments through powerful Python interfaces over a private cluster of physical hosts or a shared infrastructure (a grid, an open testbed, or a public cloud). By using containerisation and network virtualisation technologies, these tools help quasi-optimally utilise the resources of the physical infrastructure, and allow the transparent deployment of extreme-scale customised network topologies using smaller underlay structures. The virtualisation layer also allows seamless interoperability between testbeds, which further attracts users and encourages reproducibility of deployed experiments, and is ultimately a big step towards transforming computing and networking experiment platforms into scientific instruments.

However, studies [10], [11] 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 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.

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 fundamental network phenomenon –namely, the network delay– is emulated. In particular, in Section II we will first attempt 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. We will then present the main contribution of this paper in Section III: 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 present an implementation with Mininet and Distrinet. The operation of our framework will be demonstrated in a sample experiment in Section IV. The last section concludes the paper and introduces our current and future work.

II. Emulation Fidelity

A. Phenomenal Fidelity

Similar to network simulation, limited realism is one of the main challenges of network emulation. Both of these paradigms only implement simplified models and software/virtual replicas of hardware components. These models may leave out features and details which are assumed to be irrelevant but which can negatively impact the experimented scenario and ultimately alter its results. In principle, the researcher is responsible for the results that the network emulator produces, and the burden of cautiously analysing the entire emulation process falls on them. In practice, however,

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.

interpretation of emulation results is not properly conducted as filtering emulation bias out from experiment results is rarely straightforward. Hence the need for a framework to help evaluate the output.

A key concept in the evaluation of emulation accuracy is *fidelity*. Intuitively, fidelity is a measure of the quality of modeling and replication of networks and their components. It can be defined either *noumenally*, by targeting the internal, low-level behaviour of emulated components (applications, operating systems, hardware, etc.); or *phenomenally*, by considering higher-level, observable network phenomena (traffic throughput, link capacity, packet loss and delay, etc.) that can be modeled into deterministic laws. The former allows assessment of fidelity with finer granularity but is harder to measure without overhead; the latter is useful for being operational by design, as it can be concretely and systematically measured and evaluated.

This paper proposes a framework for monitoring the phenomenal fidelity of distributed network emulations. In particular, the proposed methodology examines packet delay as a candidate phenomenon for fidelity monitoring. The choice of such measure is motivated by its many outstanding properties. Indeed, the delay as a specific metric satisfies all the necessary properties required from a good measure of fidelity:

- It is scenario-agnostic and measurable in all network experiments regardless of running protocols and applications;
- It allows fine-grained analysis of the network;
- It is directly impacted by the aforementioned emulation issues and can thus easily reflect these problems otherwise invisible to the user;
- It can be efficiently measured and monitored;
- It can offer insights about the other link parameters; *and*
- It can be compared against a reference model to evaluate how well it was simulated.

In fact, it is known [2] that the measure of network packet delays at a link can be modelled and implemented following the equation:

$$d(P) = \frac{|P|}{B} + \frac{|Q(P)|}{B} + \delta, \tag{1}$$

where $d(P)$ is the total one-way delay of a packet P on the link. The first term (transmission or serialization delay) on the right hand side of the equation reflects the duration of time needed to write the packet onto the transmission medium by the interface hardware. This delay depends on the transmission speed B of the medium and the size $|P|$ of the packet. The second term (queuing delay) reflects the duration of time that the packet will spend in the queue waiting to be transmitted. It is thus equal to the sum of the transmission delays of the previous packets waiting in the queue $Q(P)$ including the remaining time of the packet currently being transmitted. The last term (propagation delay) reflects the duration of time needed for the packet to travel as a signal on the link. Popular network emulators implement links using Linux Traffic Control queuing mechanisms that behave similarly to this model.

B. Inaccurate Delay Emulation

In this section we show three typical causes of emulation failure that can be perceived from delay monitoring. These failures are either due to an inherent design of the emulators, or to the infrastructure on which they are intended to run. The first is the overload of computing resources that has been extensively studied about Mininet in the literature; the second is Mininet and its variants overlooking by default the precise emulation of the transmission delay; and the third is the additional delay caused by the physical network infrastructure in the case of distributed emulation.

1) CPU overload: Many previous studies have focused on the impact a lack of computing resources and their contention can have on network emulators' fidelity and effectiveness [10]. To demonstrate how much an overloaded CPU affects the emulated network delay, we emulate the following network in a distributed emulator: two virtual hosts H_1, H_2 are connected by a cascade of $N > 1$ virtual switches $S_1, ..., S_N$, where all links are configured with a capacity of 1 Gbps and a one-way propagation delay of 1 ms. The virtual hosts are hosted in one physical machine, and the switches in a second. On this emulated testbed we run two scenarios: in a first scenario the virtual hosts simply exchange small size (90 bytes) ICMP echo request/reply packets at a rate of 10 packets per second; while in the second scenario H_1 also sends a heavy Iperf[1] flow to H_2. We run each of these scenarios in two different settings: in a first setting only the emulator threads and basic background kernel functionalities are running (low CPU load); while in the second setting we run CPU- and memory-intensive user processes to overload all cores of the machine hosting the virtual switches. The objective of such a design is to see how the lack of computing resources negatively impacts the emulation of networking components (switching nodes and network links) even when the applications that generate the packets (Ping and Iperf) run without concurrency from other user and kernel threads. Our performance indicator of interest is the round-trip time (RTT) reported by Ping, which corresponds to the round-trip application-level delay between virtual hosts H_1 and H_2.

Figure 1 shows the results. Under low CPU load, the RTTs linearly increase with the number of intermediate switches as each link adds 1 ms of propagation delay, as well as a relatively low queuing delay in the presence of Iperf traffic. However, when the virtual switches and the emulated links experience low amounts of available CPU resources and memory bandwidth, the results are a very high increase in reported delay, especially when parallel network traffic is emulated. These results are not unexpected, but such a large impact of resource shortage, especially on delay emulation, has not been appropriately documented in the literature.

[1]Iperf3: https://iperf.fr/

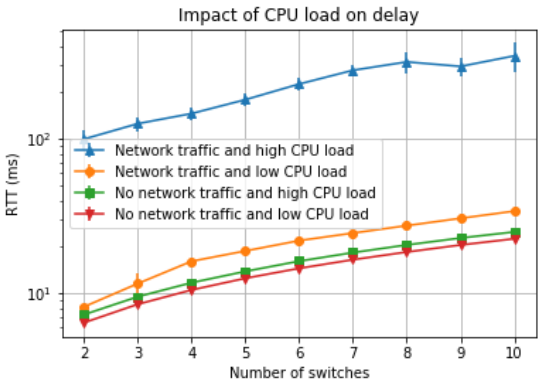


Fig. 1: Reported Ping RTT vs number of virtual switches N.

2) Non-emulation of Transmission Delay: Another source of emulation inaccuracy of Mininet and its derivatives is how they overlook correctly emulating the transmission delay of packets. Indeed, while Mininet can use TC-Netem to delay packets by a fixed (or random) value, it does not do so based on their sizes, which is the distinguishing feature of the transmission delay. Instead, as it relies on the Hierarchical Token Bucket (HTB) [4], Token Bucket Filter (TBF) [9], and Hierarchical Fair Service Curve (HFSC) [12] scheduling and queuing models, the *service time* is incurred on subsequent packets and not on the head-of-line packet itself. This error is harmless enough for the emulation of high-speed networks (less than a millisecond for regular-sized Ethernet packets in 100 Mbps and 1 Gbps links), but it can give biased results in low-bandwidth scenarios where the application-level QoS depends on the variation of the delay between packets. Thus the user must go beyond the available API to implement transmission delay if the emulations requires it.

To see this error in practice, we emulate the following networking scenario: two hosts connected to two switches that are linked through one 10 Mbps link of 1 ms propagation delay. We then simply send ICMP echo request/reply packets of different sizes (from 100 to 1500 bytes with 10 bytes increment and 1000 packets per size) at relatively large interarrival times (i.e., at a rate much lower than the link's capacity in order to avoid queuing) from one machine to the other, and log the measured round-trip delays. The result is that the measured delay does not increase with the sizes of the packets, remaining around 2ms for all pairs while it is expected to exceed 4ms for 1500-bytes echo request-response pairs of packets. In fact the Pearson correlation coefficient between size and measured RTT is less than 5%, proving that the delay is not correctly emulated.

However, it is possible to make use of mechanisms already available in TC-Netem to correctly emulate this missing delay. We propose a Mininet patch [2] to integrate this functionality. Using this patch corrects this error and we observe that the measured delay changes linearly as a function of the ICMP echoes' sizes with a coefficient of determination close to 1.

3) Underlay Network Delay: Using testbeds and platforms managed by large institutions and country- or continent-wide projects is a good opportunity for individual researchers or small research labs to run large-scale experiments or those that require expensive hardware[3]. However, multiple active users on a shared infrastructure can interfere with each other's network traffic and may bias experiment results or compromise their reproducibility. This is particularly true when a user emulates fixed-capacity links over the shared network infrastructure, and achieves lower traffic throughput due to congestion caused by other users' traffic. In such case, the underlay congestion results in abnormal increase of emulated delay due to queuing, packet loss, and retransmission. See Section IV for evidence of this phenomenon.

III. Fidelity-aware Distributed Network Emulation

To monitor the fidelity of an emulated experiment, we rely on the assumption that correct emulation of packet delays on virtual links of the topology is a criterion that must be met to ensure high levels of fidelity. In this section, we present our solution to use the delay as a phenomenal metric for emulation fidelity.

A. Concept

The basic principle can be formulated as the following criterion, which raises three important caveats regarding its implementation that we discuss individually.

Criterion. *If throughout the duration of an emulated experiment the deviation of the measured delays from the expected delays of a sample set of packets is too large, then that emulation should be considered incorrect.*

1) Passive delay measurement: First, to assess the fidelity of an emulated experiment from the network delays of emulated packets, these need to be accurately measured which can be difficult to achieve when the emulated network is distributed over multiple physical nodes. This subproblem has been tackled in [7] in which the authors have proposed to use the extended Berkeley Packet Filter (eBPF) programming framework to implement a tool that passively and accurately measures the one-way delay (OWD) [2] of packets when time synchronization can be assumed, or the round-trip delay (RTD) [3] of pairs of packets –simply defined as the sum of their individual OWDs– otherwise. To offer a solution that is operational in big geographically distributed testbeds, we will settle for the measurement of the RTD of pairs of packets P_1, P_2 which is to be compared against the sum of their own delays $d(P_1) + d(P_2)$ expected from a correct emulation.

[2]https://github.com/distrinet-hifi/mininet

[3]SLICES is one such European-wide consortium that gathers many institutions to provide a versatile research infrastructure. Our fidelity monitoring framework has been evaluated using two of its platforms with perfect transparency: the scientific computing grid Grid5000 (https://www.grid5000.fr/) and the wireless and mobile networking testbed R2Lab (r2lab.inria.fr/).

2) Packet sampling: Second, 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, 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 [6] 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.

3) Statistical metrics: Third, the deviation between measured and expected packet delays can be evaluated using different statistical metrics. A straightforward approach is to consider the mean absolute error as a measure of deviation between measured values ($\widehat{RTD}(P_1, P_2)$) and expected values ($d(P_1) + d(P_2)$). The emulation can then be considered incorrect if the mean absolute error (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 by measuring the deviation relative to the expected 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 for 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 system, we speculate in terms of quantiles: the emulation will be considered 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.

B. Implementation

Our fidelity monitoring framework is currently implemented both as a plug-in to Mininet and Distrinet (Distrinet-HiFi[4]) and as a standalone lightweight distributed emulator (HifiNet[5]). The former implementation demonstrates its compatibility with existing emulators; the latter is part of a larger project for extreme-scale network emulation aimed at outperforming state-of-the-art tools in terms of scalability. Independent of the emulator, the proposed framework can be implemented following a distributed leader-followers architecture.

Packet loggers: are instances of a program written in eBPF specification and which therefore run in the kernel space of each physical machine in the infrastructure. Their goal is to capture and log information about sampled packets in persistent storage. 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 (timestamp, packet size, and the length of the queue at the packet's arrival) and dequeue event (timestamp) if it is sampled for monitoring.

Specifically, using eBPF we embed low-level instructions into the TC datapath, 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. This overhead was previously inspected in [7] where it was evaluated to add 50 microseconds of delay and 12 microseconds of jitter to each intercepted packet, which amounts on average to less than a microsecond with a 1 out of 100 random sampling strategy.

A collector/analyser: is the brain of the system. Its goal is to compile and analyse packet information collected by the packet loggers. It runs on the leader machine and achieves its goal in three steps: first, the data is collected from packet loggers as raw files and compiled into structured tables, which are then cross-examined to match information about packets distributed over multiple tables, and output a unique large table where each entry corresponds to a packet and contains all its info; then the component computes the measured and modeled delays of each packet and derives its individual delay emulation error; and finally from the computed errors the component evaluates the overall fidelity of the emulation using statistical metrics (mean or percentiles of absolute or percentage errors, sliding window of median error, etc.).

IV. Evaluation

So far we have presented the design of our delay-based monitoring framework and shown evidence of typical emulation failures from a network delay perspective. The underlying principle is that emulation failures manifest as incorrect 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 Distrinet to date (v1.2) on four nodes of

[4]https://github.com/distrinet-hifi/distrinet-hifi

[5]https://github.com/distrinet-hifi/hifinet

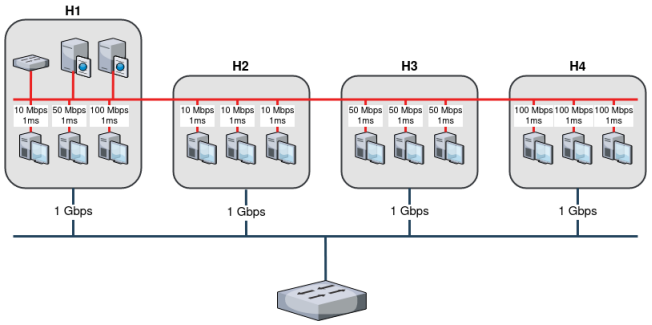


Fig. 2: Emulated network (red) and underlying cluster network. Clients from Group I are emulated in $H1$ and $H2$; from Group II in $H1$ and $H3$; and from Group III in $H1$ and $H4$.

the R2Lab cluster[6], which are connected in a star topology to one single switch (Figure 2). 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 2).

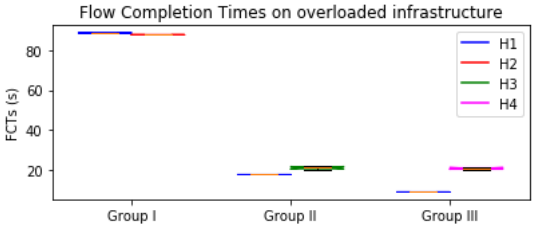
The idea is to compare the flow completion times (FCTs) and the measured delay errors between and within the groups. If this scenario were emulated with perfect fidelity (i.e., behaving exactly as it would in real networks), (a) there should be no difference in the FCTs within each group as all clients 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

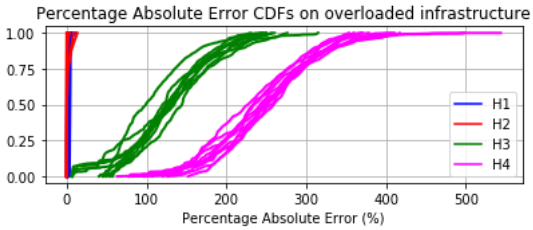
In this first run of the scenario, we emulate a total of $N = 40$ clients distributed over the 4 hosts. Figures 3 show the results. The first thing to note is how the clients in Groups II and III experience different FCTs depending on whether they are hosted in $H1$ or $H3$ and $H4$ (figure 3a). In particular, 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 3b).

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 (cf. Section II) that emulated packets experience as they cross the

[6]Reproducible Research Lab: https://r2lab.inria.fr/index.md.



(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).



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

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 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 $H1$. 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.

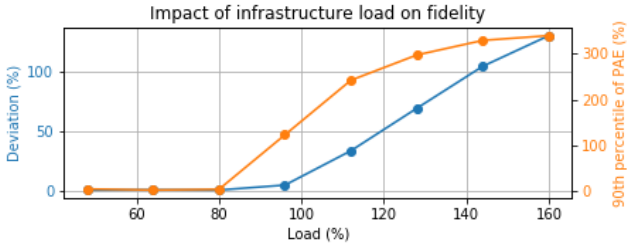


Fig. 4: High-level (blue) and low-level (orange) indicators of emulation fidelity (y-axis) vs. link load (x-axis).

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 load[7] 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 4 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.

V. Conclusion

Fidelity monitoring is essential in emulation-based experimentation to ensure certain guarantees on accuracy. A good measure of fidelity is how the finest, most elementary network phenomenon is emulated: the packet delay. We have presented in this paper an approach to fidelity monitoring of network emulation by passively measuring delays of packets in emulated links and comparing them to values estimated based on a simple network delay model. We have used the extended Berkeley Packet Filter's (*eBPF*) native packet monitoring capabilities to implement our methodology in an accurate and efficient manner. This implementation, together with a good sampling strategy, can highly limit the impact of monitoring on the emulation itself. We have demonstrated how our methodology can help predict emulation anomalies that are otherwise indistinguishable to the user from normal network behaviour. Our current implementation, alongside data to reproduce the results and scripts to reproduce the data are available at https://github.com/distrinet-hifi/distrinet-hifi. We have also briefly discussed potential sources of emulation inaccuracy inherent to the emulators and the hardware in which they are run. Our current and future work aim to provide an in-depth analysis of these –and eventually other– causes, in order to improve network emulators and their use to produce more accurate results. The passively collected delay measurements can help troubleshoot the emulation failures and accurately diagnose their causes. The use of network delay tomography can help translate the measurements collected at the emulation-level into information about the underlying infrastructure. This scheme is currently being investigated to enhance our fidelity monitoring methodology with a troubleshooting step to help the user identify the root causes of emulator inaccuracy and eventually offer solutions or workarounds.

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

References

[1] Mininet cluster edition, 2016.
[2] Guy Almes, Sunil Kalidindi, and Matthew Zekauskas. A one-way delay metric for ippm. Technical report, 1999.
[3] Guy Almes, Sunil Kalidindi, and Matthew Zekauskas. A round-trip delay metric for ippm. Technical report, 1999.
[4] Martin Devara. Hierarchical token bucket, 2003.
[5] 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.
[6] Nick G Duffield and Matthias Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM transactions on networking*, 9(3):280–292, 2001.
[7] 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.
[8] Brandon Heller. *Reproducible network research with high-fidelity emulation*. Stanford University, 2013.
[9] Alexey. N Kuznetsov. Traffic control: Token bucket filter.
[10] 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.
[11] 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.
[12] Ion Stoica, Hui Zhang, and TS Eugene Ng. A hierarchical fair service curve algorithm for link-sharing, real-time and priority services. *ACM SIGCOMM Computer Communication Review*, 27(4):249–262, 1997.
[13] 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.