Demo paper : Fidelity-aware Distributed Network Emulation

Houssam Elbouanani, Chadi Barakat, Thierry Turletti, Walid Dabbous

Fidelity-aware Distributed Network Emulation

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

***Abstract*—The design and development of new network protocols, architectures, and technologies require an evaluation phase where the researcher must provide empirical evidence for the performance of their contributions. 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 software. However, as emulation requires running multiple concurrent virtual network components on a limited amount of resources, it faces well-documented limitations in its scalability and accuracy. Our work is an attempt to address these issues by developing a lightweight and highly scalable distributed emulator, and by designing a framework for fidelity monitoring through passive measurement of emulated packet delays. The preliminary evaluation shows promising results and demonstrates the high potential of our approach.**

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

I. Introduction

Network emulators provide contained, customisable, and reproducible testing environments for researchers to evaluate their contributions. Popular network emulators (e.g., Mininet [5]) achieve this by using lightweight containers and virtual switches, and by providing users with easy-to-use APIs which let them run complex network topologies and experiments with few lines of code. In practice, many algorithms, protocols, and applications have been evaluated using this paradigm. However, recent works [6] have identified and extensively studied the inherent limitations of such testing tools, which typically emerge when large networks are emulated on a single machine with limited resources.

Recent efforts have tried to address scalability issues by proposing distributed emulation [1]: instead of running their experiment on a single machine, the user can deploy it on a private cluster of connected hosts or a public cloud. In theory, this solution allegedly overcomes any scale limitations and can let users run arbitrarily large emulated experiments, provided they can acquire enough physical hosts to sustain them. In practice, however, distribution introduces a new layer which can be a source of emulation inaccuracy: the underlay network.

This paper summarises our approach to theorise emulation fidelity and to develop a framework for fidelity monitoring, whose ultimate objective is to help users run reliable experiments with accurate results. In particular, we propose a methodology for fidelity monitoring through passive delay measurement that can detect emulation inaccuracy and potentially infer its root causes.

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.

II. High-fidelity Emulation

A. Methodology

Our contributions revolve around the key concept of *emulation fidelity*, which intuitively signifies realism and measures the similarity of an emulation to its target environment, i.e., if it were implemented using actual hardware. We specifically propose to define emulation fidelity *phenomenally*: by focusing on network phenomena (aspects that are observable, measurable, and which can be modeled into deterministic laws) instead of conceptualising fidelity at the level of internal, low-level functioning of the emulated network. Such definition is particularly useful for being both conceptual (i.e., it captures the essence of the concept and its relationship to other concepts) and operational (i.e., it can be concretely and systematically measured and evaluated) by design. Our main hypothesis is therefore that phenomenal fidelity carries information about the quality of emulation and can infer the accuracy of its results.

In particular, the proposed methodology examines the individual packet delay as a candidate network phenomenon for fidelity monitoring. The choice of this measure is motivated by many outstanding properties compared to other phenomena (flow throughput, application-level QoS, etc.). Indeed, the delay:

- is scenario-agnostic and can be systematically monitored in all experiments regardless of the running applications;
- can be easily and efficiently measured;
- allows fine-grained analysis of the network;
- is directly impacted by network-affecting anomalies;
- offers insights about other network phenomena (jitter, bandwidth, packet loss);
- can be compared to a reference model to evaluate how well it is simulated.

In fact, it is known [2] that the measure of network delay $d(P)$ of a packet P on a link can be modeled and implemented according to the formula:

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

where δ is a fixed propagation delay, B is the bandwidth of the link configured in the emulator by the user, and $Q(P)$ is the size of the link queue (including the head-of-line packet) when the packet P has entered. Thus, a judgement about the overall quality of emulation can be made by looking at how well the packet delays were simulated. In practice, this can be evaluated

by implementing a monitor that collects information each time a packet passes through a link: its timestamps of transmission $t_1(P)$ and reception $t_2(P)$, its size $|P|$, and the size of the queue at its arrival $|Q(P)|$. Then the delay emulation errors can be calculated as the absolute difference between measured and modeled delays:

$$\epsilon(P) = |\big(t_2(P) - t_1(P)\big) - d(P)|.$$

In cases where strong time synchronisation cannot be achieved, the round-trip delay of any pair of sent and received packets can instead be evaluated.

Our methodology can be summarised as follows:

If, throughout the duration of an emulated experiment, the delay emulation errors are too large, then that emulation should be considered incorrect.

III. Prototype

Our fidelity monitoring framework is currently implemented both as a plug-in to Mininet and Distrinet [3] (Distrinet-HiFi) and as a standalone lightweight distributed emulator (HifiNet). 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. This section briefly summarises the design principles and broad architecture of our framework, independent of the implementation.

A. Architecture

The design of our framework follows a leader/follower architecture typical to distributed systems, and to distributed network emulators in particular, where a leader machine manages the emulated network and allocates its components (end-hosts, switches and routers, links, etc.) to the follower machines who host them. Such distributed design complies perfectly with their architectures and allows seamless integration. Another feature of distributed architectures is their scale capabilities, which is an important requirement for a monitoring system. As such, the logic of the framework is divided into two main components that perform the passive delay measurement task at different levels.

a) Packet loggers: The packet loggers are lightweight and optimised programs that are plugged into both ends of every link that needs to be monitored. They run as kernel code in every follower machine of the cluster and their only task is to intercept packets and log information about their transmission (size, length of the queue when the packet entered the queuing system, timestamps of entry and transmission) and reception (timestamp of reception) events. This captured data is useful to passively measure the delay of emulated data packets, and also to estimate their theoretical delay from the previous model (equation 1).

b) Collector/Analyser: This component 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.).

B. Overhead and Scalability

By virtue of its design, our fidelity monitoring framework has a strong potential for scalability, the only potential overhead is the one incurred by the packet logging task. Indeed, as emulated packets are to be intercepted and their metadata logged online, this can lower the packet processing and switching performance. However, we have identified a low-overhead implementation opportunity in the Extended Berkeley Packet Filter (eBPF), which as a kernel programming framework for running user-written code in the kernel space of Linux systems, and which is specifically tailored for line-speed kernel monitoring and packet processing tools. Our previous paper [4] demonstrates in full details the capabilities of such framework for passive delay measurement, and proves to be particularly efficient as it does not incur more than sub-microsecond delay on intercepted packets.

In addition to efficient implementation, the overhead of the fidelity monitoring system can be mitigated by packet sampling. Indeed, it is not necessary to measure the delay of each individual emulated packet. Instead one can implement a random sampling strategy for large-size networks and/or heavy traffic. Our preliminary investigations indicate that a 1 out of 100 packet sampling rate yields results with enough statistical significance and near-zero overhead.

IV. Demonstration

A. Testbed

This presented series of experiments has been conducted and can be reproduced on the Grid5000[1] open experiment platform. 1 leader and 10 follower machines are used to run an emulated network of 110 SDN switches and 1000 end-hosts. This emulated network is divided into 10 ASes, each of which manages 100 end-hosts through 1 core and 10 access switches. The emulator distributes the load to the follower machines fairly: each AS is embedded in a dedicated physical node.

In this experiment, the emulated end-hosts are uniformly divided into 500 clients and 500 servers, and each client is randomly assigned a unique server from which it synchronously downloads a 10 MB file with an access bandwidth of 10 Mbps. The experiment is stopped once every client has finished downloading. In parallel, our fidelity monitoring framework

[1]The Rennes cluster was used to run the experiments. More details about the topology, system and hardware specifications can be found at https://www.grid5000.fr/w/Rennes:Hardware

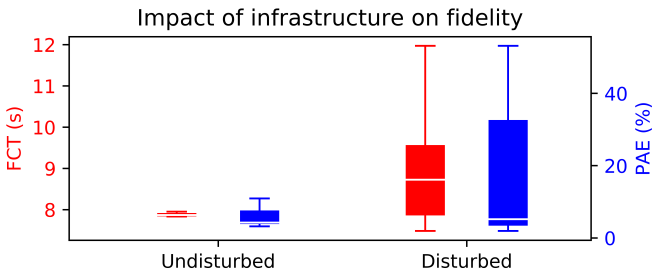

Fig. 1: High-level (red) and low-level (blue) of emulation fidelity in an undisturbed (left) and disturbed (right) infrastructure.

passively measures the delay emulation errors of each virtual link whose ends cross the infrastructure network (i.e., inter-core links), and is configured with a 1% random sampling rate. The source code of the fidelity-enhanced emulator and instructions to set-up and reproduce the experiment can be found at https://github.com/distrinet-hifi/hifinet. A python notebook along with experiment data is also available to reproduce the presented results.

B. Results

As both the overlay and underlay networks are over-dimensioned and the embedding optimal, congestion is not to be expected and thus each download session should be expected to last approximately 8 seconds (10 MB / 10 Mbps). Figure 1 effectively proves this where we observe that the flow completion times (FCTs) are indeed in an narrow interval around 8 seconds. In parallel, our fidelity monitoring framework does not report particularly large delay emulation errors: the 95th percentile of percentage absolute error (PAE) is less than 10%, which means that at least 95% of all sampled packets were not delayed by more than 10% of their expected delays, indicating sound emulation.

However, in a second set of runs an external traffic is generated in the infrastructure to introduce disturbance, which simulates traffic from other users in the shared underlay infrastructure. The results change and we observe increases in the FCTs: the average, standard deviation, and maximum increase from 7.90s, 0.17s, and 9.91s, respectively, to 9.05s, 1.31s, and 14.35s. In parallel, the fidelity monitoring system reports increases in the PAE in some inter-AS links to 53.12% of delay emulation error.

C. Discussion

The flow completion times of downloads increase when external traffic in the underlay infrastructure causes congestion which impacts the correct operation of the emulated network. These results demonstrate that the high-level results of an emulated experiment depend on the state of the infrastructure, which the user might not have complete awareness of and/or control over (as is the case in shared infrastructures such as the used experiment platform). And unless a careful modelling and analysis of the emulated scenario is conducted, it is difficult to determine whether the obtained emulation results were corrupted by underlying issues, and impossible to localise the failures. The monitoring of the emulated delay helps solve this problem by providing systematic information about the fidelity of the emulation, and can even help troubleshoot its failure. In this scenario, for instance, increase in delay emulation error has occurred in a specific set of inter-AS links only, which can already help rule out certain infrastructure machines and links.

D. Reproducibility

We will repeat the steps of the presented experiment to showcase how our fidelity-enhanced emulator works in practice, and how well it performs to evaluate the fidelity of the emulation under different settings (undisturbed infrastructure, disturbed infrastructure with different traffic profiles, etc.). The live demo will also display the results with more granularity: the increases in completion times of flows from clients in specific ASes to servers in other ASes, and the measured delay emulation error in individual inter-AS links. The code to produce and reproduce these results is available as a Python notebook at https://github.com/distrinet-hifi/hifinet.

V. Current and Future Work

Our fidelity monitoring framework has proven to be efficient for file download scenarios involving average-size clusters of machines. We are currently investigating its viability for more complex emulated scenarios –where fidelity is necessary to detect biased emulation results– and larger infrastructures –where troubleshooting bottleneck links is not straightforward. This will be tested by a larger-scale series of experiments where we induce controlled infrastructure issues and assess how precise our tool is in identifying them. We are also examining how the framework can learn from the troubleshooting phase to help the user better revise the emulated experiment.

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] 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.
[4] 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.
[5] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.
[6] 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.