MSTG: A Flexible and Scalable Microservices Infrastructure Generator

Emilien Wansart
Montefiore Institute
Université de Liège
Belgium
emilien.wansart@uliege.be

Maxime Goffart
Montefiore Institute
Université de Liège
Belgium
maxime.goffart@uliege.be

Justin Iurman
Montefiore Institute
Université de Liège
Belgium
justin.iurman@uliege.be

Benoit Donnet
Montefiore Institute
Université de Liège
Belgium
benoit.donnet@uliege.be

***Abstract*—The last few years in the software engineering field have seen a paradigm shift from monolithic applications towards architectures in which the application is split in various smaller entities (i.e., microservices) fueled by the improved availability and ease of use of containers technologies such as `Docker` and `Kubernetes`. Those microservices communicate with each other using networking technologies in place of function calls in traditional monolithic software. In order to be able to evaluate the potential, the modularity, and the scalability of this new approach, many tools, such as microservices benchmarking, have been developed with that objective in mind. Unfortunately, many of these tend to focus only on the application layer while not taking the underlying networking infrastructure into consideration, leading to difficulties in developing and testing telemetry tools.**

In this paper, we introduce and evaluate the performance of a new modular and scalable tool, *MicroServices Topology Generator* (MSTG), that allows one to simulate both the application and networking layers of a microservices architecture. Based on a topology described in YAML format, MSTG generates the configuration file(s) for deploying the architecture on either `Docker Compose` or `Kubernetes`. Furthermore, MSTG encompasses telemetry tools, such as Application Performance Monitoring (APM) relying on `OpenTelemetry` and in-band telemetry (e.g., IOAM). This paper also discusses a use case in whic MSTG finds a suitable usage.

I. INTRODUCTION

Modern cloud-native applications rely on microservices. A single request in an application can invoke a lot of microservices interacting with each other over the network. Consequently, it is becoming increasingly difficult to monitor and isolate a problem. This is why Application Performance Monitoring (APM), based on distributed tracing tools, e.g., `OpenTelemetry` [1] combined with `Jaeger` [2]), is useful. APM provides a way to observe and understand a whole chain of events in a complex interaction between microservices. However, such an APM appears as useless when the problem is not application related but rather located at the network level. Therefore, there is a need for new telemetry tools that would be able to monitor both the microservices and the network layer underneath them. Prior to be developed, such tools must be carefully evaluated in a controlled environment to ensure their accuracy, efficiency, and that they will not disturb the various microservices. However, existing microservices benchmark generators [3], [4], [5], [6] totally ignore network resources (e.g., routers) required to run microservices in the cloud and are not dedicated to test integrated telemetry solutions for microservices.

978-3-903176-64-5

In this paper, we introduce the ***M**icro**S**ervices **T**opology **G**enerator* (MSTG), an application able to generate a complete microservice topology that includes both network components, such as routers, and microservices. In particular, this paper makes the following contributions:

- we carefully describe MSTG (Sec. II). In a nutshell, MSTG simulates microservices and network architecture through `Docker` based on a topology described in a YAML configuration file. The resulting topology can be deployed on a single machine with `Docker Compose` [7] or distributed with `Kubernetes` [8];
- we evaluate the performance of MSTG and show it is flexible and scalable (Sec. III);
- we demonstrate, through a use case (Sec. IV), a suitable usage of MSTG for evaluating the impact of telemetry tools in microservices infrastructures;
- finally, our source code and a technical report [9], which contains more details about MSTG, are made available to the community.

II. MICROSERVICES TOPOLOGY GENERATOR (MSTG)

The ***M**icro**S**ervices **T**opology **G**enerator* (MSTG) facilitates the simulation of microservices architectures through container technologies. It relies on a configuration file to build a topology, which is a structure of microservices interconnected by routers. Those microservices can be parameterized at both the application and network layers.

The primary objective of this tool is to offer a customizable environment for demonstration and testing purposes. Simulating scalable microservice topologies proves valuable in various scenarios such as validating the correctness of an architecture, conducting testing and benchmarking of a topology, and assisting in the evaluation and integration of other technologies into a microservices architecture, such as telemetry or monitoring tools, before their inclusion into a critical production environment on which a potentially non-negligible number of users are relying.

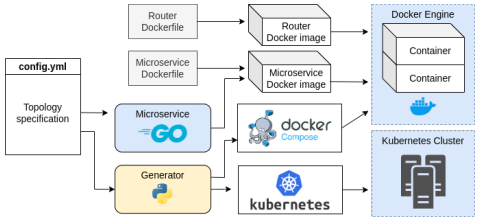


Fig. 1. MSTG architecture.

A. Architecture and Components

MSTG architecture, depicted in Fig. 1, can be described as follows. The configuration file, *config.yml*, serves as the tool input, containing the topology structure with interconnected microservices and, optionally, routers with configuration of the networking layer properties. The *generator* parses the configuration file and generates the necessary file(s) for deploying on either `Docker Compose` or `Kubernetes`. The *microservice* is the application (Layer 7) that listens to `HTTP` or `HTTPS` requests and responds to them following the sequential query of other services based on the configuration file. It can run over IPV4 or IPV6 and perform telemetry using `OpenTelemetry` and sending them to `Jaeger` [2] for storage and visualization. The *routers* (Layer 3) that interconnect the microservices by forwarding their traffic. The ability to include routers enriches the topology with an IP layer, enabling the use of lower-layer monitoring tools such as IOAM for in-band network telemetry [10], and to have an architecture closer to what one would find in a datacenter network (DCN) or a cloud network. Finally, the topology can be deployed on either `Docker Compose` or `Kubernetes`.

The generation process offers options, including the choice to enable microservices tracing based on `OpenTelemetry` [1] with `Jaeger` [2], and in-band telemetry with IOAM [10] (IOAM gathers telemetry and operational information along a path, within IPV6 header of packets). Those solutions are the basis for developing advanced telemetry tools for microservices infrastructures.

III. EVALUATION

Our methodology is structured to comprehensively evaluate MSTG capabilities. It involves two main aspects: firstly, ensuring the correctness of the tool's functionality, where input variables are accurately reflected in the generated topologies, and secondly, evaluating the tool's performance.

Unless specified otherwise, all measurements are averages obtained over ten samples collected over a 30-second interval during 10 distinct executions of the tests to ensure representativeness of the statistics while the shaded areas around the curves are standard deviations. All the experiments were conducted using `HTTP` for communication between the microservices to have measurements for the most basic case (i.e., `HTTP`) without the overhead introduced by `TLS`. The experiments were conducted on a server equipped with an Intel Xeon CPU E5-2630 v3, 8 cores, 16 threads, 32GB of RAM, and running Linux 5.17.

The ability to simulate networks having real-world characteristics is dependent on `Docker` and the Linux Kernel since MSTG relies on them. To verify the correctness of MSTG, we focus on the following key configuration options since they have substantial performance impacts that can be assessed by performance measurements: microservice packet size, router link delay, and loss rate. We selected the values of the variables presented hereafter in order to cover real deployments of microservices architectures.

As shown in Fig. 2a, the graph represents the influence of the packet size on the maximum network data input (RX) and output (TX) for a simple topology with two microservices connected via a router. As depicted, the network usage is increasing with the packets' size, which is the expected result.

We assess the impact of the link delay on the round-trip time (RTT) and the effect of loss rate on the maximum request rate attainable, as shown in Fig. 2b and 2c, with the topology being identical to the previous experiment. Due to the relation between these variables, we can conclude that MSTG is rightly simulating a network.

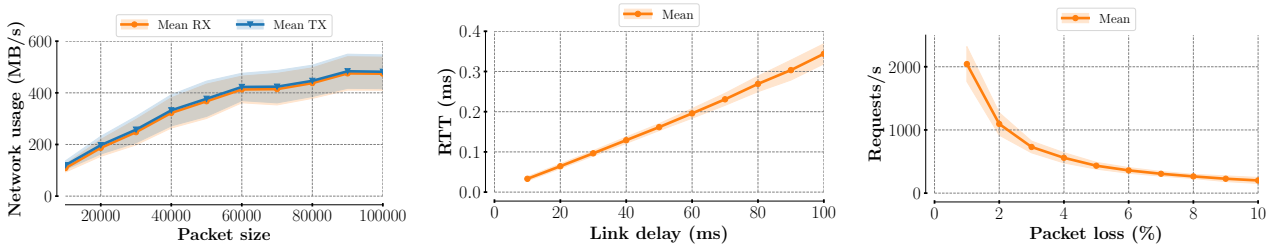
The times to generate, start, and stop an architecture are all proportional to the size of the topology. Based on our evaluation, for an architecture compose of 100 entities, it takes, on average over 100 executions, 85ms to generate it and 12s to start or stop it while deploying it on `Docker Compose` and the aforementioned processor.

IV. HAPROXY USE CASE

MSTG can be used to assess the feasibility of implementing an intelligent microservice selection mechanism. Usually, in a microservices architecture, there are many instances of the same service to increase the availability and provide the best experience to the end users. In such a case, the instances are behind a proxy, which is acting as a load-balancer, that redirects each request to one of the available instances based on some predefined criteria. This section describes a particular scenario that combines in-band telemetry (i.e., IOAM [10]) with a load balancer, `HAProxy` [11].

As illustrated in Fig. 3, a client reaches the microservice instances through a transparent proxy. The load-balancer is responsible for sending packets augmented with IOAM data, which will contain the queue depth of every router along the path within the datacenter, to every instance at regular time intervals. Once the instances receive the packets, they extract the IOAM data, gathered along the hops (routers) between them and the proxy, and send it to a centralized controller. The controller will reconfigure the proxy by adjusting the distribution of the packets proportionally to the metrics observed on the paths and collected by IOAM.

We evaluated this new microservice selection solution by using MSTG to generate the architecture depicted in Fig. 3 and measured the percentage of the traffic propagated to each instance over time. At time T=20S, we generated a load between the load-balancer and the first instance, while at T=40S, we generate a load twice as high between `HAProxy` and the second instance. As observable in Fig. 4, a few



(a) Maximum network utilization (RX & TX) per packet size.

(b) RTT per link delay.

(c) Maximum request rate per loss rate.

Fig. 2. MSTG evaluation.

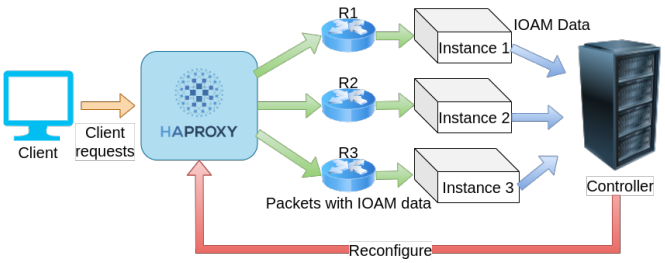


Fig. 3. Architecture of service selection with HAProxy.

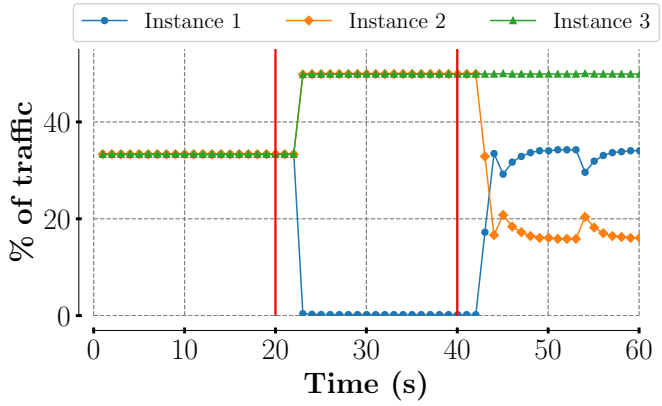


Fig. 4. Service selection with `HAProxy`.

seconds after the loads are generated, the collector computes the new proportion of the traffic forwarded to each instance and reconfigures `HAProxy` with these new values. Thus, the instances on overloaded paths get less of the subsequent traffic based on the IOAM metadata.

V. Conclusion

Microservices architectures are becoming the default paradigm and can replace monolithic applications in most scenarios. Yet, most tools to evaluate different criteria about this new approach give too few considerations to the networking layer, which is a critical component for this design pattern.

This paper proposed and evaluated ***Micro**Services **T**opology **G**enerator* (MSTG), a microservices topology generator that gives back to the networking layer the attention it deserves by creating a modular and scalable microservices topology generator. MSTG offers the possibility to simulate both the networking and application layers, which are configurable by the end-users in a configuration file used as input. As demonstrated by empirical measurements, MSTG provides to users the ability to easily and rapidly test their microservices architectures and their integration with other telemetry or monitoring technologies before deploying them on a production environment on which a significant amount of customers may depend.

MSTG is an ongoing project. Future works would consider to add support for switches in the generated topologies, in order to be even closer to real deployments, and to improve the user experience of MSTG.

Software Artefact & Acknowledgments

Source code of MSTG is available at https://github.com/Advanced-Observability/Micro-Services-Topology-Generator. A technical report of MSTG is available [9] and contains more details about the inner-workings of MSTG, a more thorough evaluation, and some interesting use cases.

This work is supported by the CyberExcellence project funded by the Walloon Region of Belgium, under number 2110186, and the Feder CyberGalaxia project.

References

[1] OpenTelemetry, "Effective observability requires high-quality telemetry," see https://opentelemetry.io.

[2] Jaeger, "Open-source, end-to-end distributed tracing," see https://www.jaegertracing.io.

[3] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, April 2019.

[4] M. R. Saleh Sedghpour, A. Obeso Duque, X. Cai, B. Skubic, E. Elmroth, C. Klein, and J. Tordsson, "HydraGen: A microservice benchmark geneartor," in *Proc. IEEE International Conference on Cloud Computing (CLOUD)*, July 2023.

[5] A. Sriraman and T. F. Wenisch, "υ-suite: A benchmark suite for microservices," in *Proc. IEEE International Symposium on Workload Characterization (IISWC)*, September 2018.

[6] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2012.

[7] Docker, "Docker compose overview," [Last Accessed: January 23th, 2024]. [Online]. Available: https://docs.docker.com/compose/

[8] Kubernetes, "Production-grade container orchestration," [Last Accessed: January 23th, 2024]. [Online]. Available: https://kubernetes.io/

[9] E. Wansart, M. Goffart, J. Iurman, and B. Donnet, "MSTG: A flexible and scalable microservices infrastructure generator," arXiv, cs.NI 2404.13665, April 2024.

[10] F. Brockners, S. Bhandari, and T. Mizrahi, "Data fields for in situ operations, administrations, and maintenance (IOAM)," Internet Engineering Task Force, RFC 9197, May 2022.

[11] HAProxy, "The reliable, high performance TCP/HTTP load balancer," [Last Accessed: January 30th, 2024]. [Online]. Available: https://www.haproxy.org/