Lightweight Self-adaptive Cloud-IoT Monitoring across Fed4FIRE+ Testbeds

Marco Gaglianese, Stefano Forti, Federica Paganelli, Antonio Brogi
Department of Computer Science, University of Pisa, Pisa, Italy

***Abstract*—Monitoring Fog infrastructure resources in a lightweight and fault-resilient manner is a challenging research problem. In this article, we illustrate the experimental assessment of a distributed, self-organising and fault-tolerant monitoring tool especially targeting Fog environments. The assessment involved up to 40 nodes across two testbeds within the Fed4Fire+ federation. Results show the capability of the tool to handle different types of failures in the monitored infrastructure, and to quantify its measurement accuracy and limited footprint.**

I. Introduction

Monitoring Fog resources [1] will be crucial to orchestrate next-gen services. Indeed, monitoring data are required to decide where to deploy application services, and when and where to scale and migrate them in case their Quality of Service (QoS) and contextual requirements cannot be satisfied by the current deployment and infrastructure state [2]. However, monitoring is especially challenging to implement due to various peculiarities of Cloud-IoT settings, e.g., limited hardware resources and unstable connectivity at the Edge, platform heterogeneity, and node or connection failures. Therefore, experimenting Cloud-IoT monitoring solutions in real systems or testbeds is essential for their validation.

In this context, the *Lightweight Self-adaptive Cloud-IoT Monitoring across Fed4FIRE+ Testbeds* (LiSCIo) experiment[1] aimed at evaluating over a distributed testbed environment the operation of a monitoring service, called FogMon, targeting heterogeneous Cloud-IoT environments. Such an experiment run over a distributed testbed within the Fed4Fire+ federation.

FogMon monitors hardware resources at Cloud-IoT computing nodes and end-to-end network QoS between such nodes [3]. FogMon features a self-organising peer-to-peer overlay topology with self-restructuring mechanisms and differential monitoring updates to ensure fault-tolerance and scalability. While various tools are available for monitoring Cloud infrastructure resources, only a few prototypes have been proposed for monitoring node resources (like [4]) or node resources and end-to-end latencies (like [5]) in Fog environments. As thoroughly discussed in [3], FogMon is the first monitoring tool capable of measuring and aggregating in a scalable, robust, and non-intrusive manner information on hardware resources, end-to-end network QoS parameters, and available IoT devices in Fog environments.

Work partly supported by experiment *Lightweight Self-adaptive Cloud-IoT Monitoring across Fed4FIRE+ Testbeds* (F4Fp-08-M30) funded by Fed4Fire+ (Horizon 2020, G.A. 732638). Experiments run from November 2020 to April 2021. More information at https://www.fed4fire.eu/demo-stories/oc8/liscio/

[1] *Fed4Fire+*, https://www.fed4fire.eu

The objective of LiSCIo was to test, assess and tune FogMon over a distributed infrastructure of Cloud and Edge nodes built by leveraging the VirtualWall and CityLab testbeds of Fed4Fire+. The experiment setup consisted of a Docker-based deployment of FogMon over a set of heterogeneous nodes, i.e., 10 CityLab wireless nodes and 10-30 VirtualWall servers. Ultimately, LiSCIo led to implementing, assessing and releasing a new version of FogMon, called FogMon2, which improves (*i*) the *relative error* on the latency and bandwidth values measured with respect to a setup ground truth, and (*ii*) the *time to stability*, i.e., the time needed to reach a stable overlay organisation after initialisation or after node failures or link QoS degradations. Overall, FogMon2 upgrades the technology readiness level (TRL) of our prototype from TRL4 (validation in a laboratory environment, 13 nodes) to TRL5 (validation in a relevant environment, 40 nodes).

In this article, we focus on FogMon2, illustrating its experimental assessment over the federated testbeds of Fed4Fire+. Due to space limits, we do not describe here the preliminary experiment activities conducted on the previous version (FogMon), which allowed us to fix some issues and release FogMon2. We run and discuss two main types of tests: (*i*) node failures and (*ii*) link failures, at different severity levels. We analyse the capability of FogMon2 to self-restructure so to cope with such failures/degradations. We also consider two different configurations of FogMon2 – a default and a reactive one – to evaluate the overhead due to improving FogMon2's reactiveness to failures. To this end, in all tests, we collect the footprint of FogMon2 on CPU, RAM, and bandwidth.

The rest of this article is organised as follows. Sect.II presents FogMon2, Sect.III and IV describe setup and results of the LiSCIo experiment, and Sect.V draws some conclusions.

II. Fog infrastructure monitoring with FogMon2

FogMon2 is a C++ open-source[2] software prototype implemented and released throughout the LiSCIo experiment. It provides distributed monitoring features targeting Fog computing environments, consolidating and improving the original FogMon methodology and prototype [3], [6]. FogMon2 monitors hardware resources (viz., CPU, memory, hard disk) at different Fog nodes and end-to-end network QoS (viz. latency, bandwidth) between such nodes. FogMon2 nodes organise themselves into a peer-to-peer overlay network that features two types of software agents: *Followers* and *Leaders*.

[2] https://github.com/di-unipi-socc/FogMon/tree/liscio-2.0

Follower nodes probe all monitored metrics and are divided into groups, each group being associated with a *Leader* node. They can leave and join the network at any moment in time, due to their own decision as well as to node or network failures. *Leader* nodes, besides monitoring data about their own deployment node and other *Leaders*, periodically aggregate data gathered from their *Follower* nodes. *Leaders* self-organise into an overlay peer to-peer network and share data aggregated from their *Followers* through gossiping.

The topology of the FogMon2 monitoring overlay is constructed upon a *proximity criterion* based on latency measurements among nodes. We assume that any new node joining the FogMon2 environment initially acts as a *Follower*, which only knows the address of one (or some) *Leader* node(s), deployed at a known location. Such a node acts as a registry of all *Leader* identifiers (viz., IP addresses and ports).

Any new *Follower* retrieves from the known *Leader* a list of those identifiers. Subsequently, it measures the latency (i.e., the round-trip time) against each of them and it associates with the group of the closest *Leader*. The *Follower* periodically checks the network status and it may change *Leader* when it cannot reach the associated *Leader* anymore due to network or node failures, or when it identifies a closer candidate *Leader* (e.g., due to network congestion, or newly available *Leaders*).

To tame the quadratic time complexity of measuring end-to-end QoS in a network of N nodes, the number of clusters (i.e., *Leaders*) is set to $\sqrt{N}$ by default. After joining the network, *Follower* nodes can start their normal probing operation of hardware resources and intra-group end-to-end QoS, periodically reporting monitored data to their *Leader*. To reduce communication overhead, *Followers* only send data whose average or variance differ more than a given threshold (i.e., 10% by default) from the last performed update.

While network QoS is actually measured within each group and between *Leaders*, latency and bandwidth between nodes in distinct groups are estimated by *Leaders* relying on gossiped data. More specifically, latency $\ell_{A,B}$ between two *Followers* A and B referring to distinct *Leaders* $L1$ and $L2$ respectively, is computed as: $\ell_{A,B} = \ell_{A,L1} + \ell_{L1,L2} + \ell_{L2,B}$. This approximation is acceptable as long as the latency between *Leaders* is greater than the latency between each *Follower* node and its *Leader*. The assumption is reasonable as per the proximity criterion used to build up and maintain the FogMon2 network.

Analogously, the available bandwidth $\beta_{A,B}$ from node A to node B is approximated as the minimum between the maximum outgoing bandwidth of A and the maximum incoming bandwidth of B and the bandwidth between their *Leaders* $L1$ and $L2$: $\beta_{A,B} = \min(\max_k \beta_{A,k}, \max_h \beta_{h,B}, \beta_{L1,L2})$.

FogMon2 can handle (*i*) network disconnections and node crashes at *Followers* by removing unreachable nodes from *Leader* databases when they disconnect for a prolonged time, and (*ii*) *Leader* failures by allowing *Followers* to join the group of a new (different) *Leader* and start again their monitoring activity. Data replication through gossiping at all *Leaders* guarantees monitoring data availability.

Besides, FogMon2 topology self-restructures to cope with dynamic variations in the Fog infrastructure by exploiting a distributed execution of a *k-medoids* clustering algorithm [7] at each Leader, combined with a consensus protocol to decide on the best restructuring. A full topology restructuring mechanism is triggered when one of the following events occurs: (*a*) the network size doubles/halves from the last restructuring, or when (*b*) clustering quality measured at *Leaders* exceeds a threshold value. A topology restructuring is also triggered when the number of *Leaders* considerably differs from $\sqrt{N}$.

FogMon2's behaviour is highly configurable to allow fine-tuning the balance between being lightweight on infrastructure resources and quickly detecting changes in the monitored metrics. We consider two possible configurations. The *default* configuration favours a non-intrusive behaviour of the prototype (e.g., Follower reporting period is 30 s and bandwidth probing period is 10 minutes), while the *reactive* configuration favours monitoring promptness (e.g., Follower reporting period is 15 s and bandwidth probing period is 2 minutes).

In comparison with FogMon, FogMon2 improves some implementation details: *i)* latency and bandwidth measurements from a Follower to the others in the same group are performed in parallel by different threads (instead of being performed sequentially) to speed-up the overall time to complete all measurements; *ii)* in order to reduce the time required to detect and react to network QoS degradations, FogMon2 empties the latency (bandwidth) measurement windows in case the degradation exceeds a certain set threshold (500% for latency, 30% for bandwidth) with respect to the current average value; *iii)* passive and active bandwidth measurements performed with Assolo and iperf, respectively, are better tuned.

III. Experiment Setup

a) Testbeds: LiSCIo run over two testbeds within the Fed4Fire+ federation, viz. VirtualWall (https://www.fed4fire.eu/testbeds/virtual-wall) and CityLab (https://www.fed4fire.eu/testbeds/citylab), both managed by imec. The VirtualWall testbed (Ghent, Belgium) provides more than 550 servers can be used as bare metal hardware or virtualized through XEN or docker. VirtualWall supports multiple operating systems and software-enabled network impairment capabilities (e.g., delay, packet loss, bandwidth limitation). All nodes have a public IPv6 address. The CityLab testbed (Antwerp, Belgium) is designed for large-scale wireless networking experimentation in smart cities. The testbed is deployed around the city and the university campus, in 50 locations, and it represents a realistic wireless environment for edge computing. Nodes connect to the Internet through a gateway relying over a fibre network.

b) Experiment setup and tooling: We set-up an experimental environment that reproduces a distributed infrastructure of Cloud and Edge nodes. We jointly relied upon (T1) the CityLab testbed to be used for its Edge computing gateways, running FogMon2 from 10 to 20 locations, and (T2) the Virtualwall testbed to be used for its bare-metal servers. In VirtualWall we reserved 10+ physical nodes. In CityLab we reserved 10+ wireless nodes. All nodes run an Ubuntu 18.04 LTS distribution on top of which we installed Docker 20.10.5

to run FogMon2. We defined two infrastructure sizes based on the number of exploited nodes, viz. a *Small* infrastructure with 20 nodes (10 on VirtualWall and 10 on Citylab), and a *Large* infrastructure (30 on VirtualWall and 10 on Citylab).

On each node, FogMon2 is deployed and runs in a container by relying on the configuration and management settings which can be downloaded and extended from JFed[3]. Based on these settings and by exploiting Fabric, we implemented suitable helper methods to install Docker, to setup end-to-end network links by means of GRE tunnels and tc, to pull/start/stop FogMon2, to deploy a monitoring tool for bandwidth usage (i.e., bmon). Our LiSCIo *Topology Builder* tool:

1) creates a large tree, mimicking the hierarchical structure of the Internet with nodes closer to the root representing data centres and ISP nodes, and leaf nodes representing access points and edge devices,
2) within the graph generated at (1), it selects the number of nodes specified for the current experiment and prunes the others,
3) computes the end-to-end latency among the selected nodes by summing up latencies along paths, and it computes bandwidths among the selected nodes by taking the minimum bandwidth along paths,
4) computes the Davies-Bouldin index [8] for the selected graph to assess the overlay quality of the graph before running FogMon2 over it, and
5) based on the results of step (4), it generates an `rspec.xml` file that can be input to JFed, containing all node details.

The *Topology Builder* graphically outputs the results of the steps 1–5 as shown in Fig.1, where red nodes represent supporting routers, and other colours a possible clustering of 20 nodes in a candidate *Small* testbed. The link configuration (specified with latency and bandwidth values computed as described at point 3) is output by the *Topology Builder* and it is used as:

- input to the aforementioned Fabric helper methods to set up end-to-end links between nodes in the testbed, and
- ground truth[4] to compute FogMon2 relative errors on bandwidth and latency measurements and estimates.

An open-source[5] service, FogMonEye, is deployed to an external node to collect and visualise live experiment results as shown in Fig. 2. Such a service collects periodic reports from *Leader* nodes and analyses and compare them against the ground truth provided by the *Topology Builder* so to compute and visualise:

(a) the relative error of FogMon2 on the measured intra-group latency and bandwidth and the relative error on the estimated inter-group latency and bandwidth, computed as the relative error between the ground truth (i.e., the testbed configurations) and the values collected/computed by FogMon2,
(b) the footprint of FogMon2 on CPU, RAM (measured via docker stats) and bandwidth (measured via bmon), and
(c) the time that FogMon2 employs to reach stability[6] after infrastructure changes that trigger a topology restructuring (e.g., crash of a node, changes in link QoS).

FogMonEye permits to concurrently monitor multiple experiment sessions with FogMon2 and allows to store the results of completed experiments. The given metrics will be used in Sect. IV to discuss the experiment results.

c) Experiment types: Experiments with FogMon2 are divided into two categories:

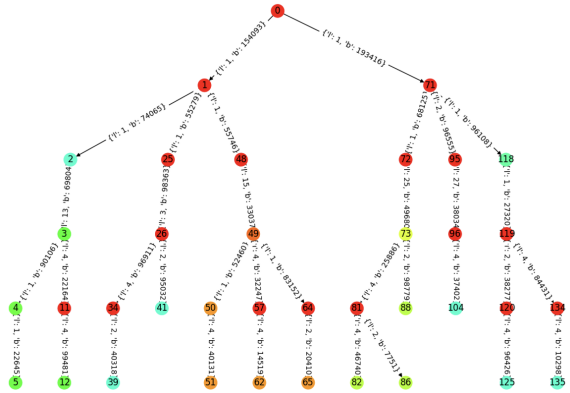


Fig. 1: *Topology Builder*: example output.

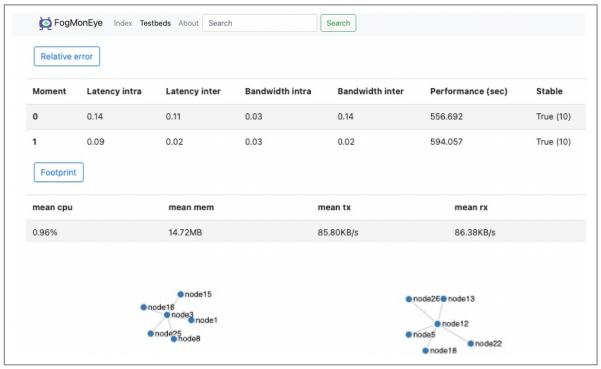


Fig. 2: FogMonEye WebGUI.

[3]*JFed* is an open-source Java-based application that represents a single point of contact and service for Fed4Fire+ testbeds. It enables a fine-grained configuration and management of experimental resources across different testbeds through the usage of standard tooling, i.e., Ansible, Fabric, and a file specification of experiment requirements, i.e., `rspec.xml`.

[4]As Fed4Fire+ testbeds enable setting up the latency and bandwidth featured by each monitored end-to-end link according to the values defined by the *Topology Builder*, we decided not to rely on any external monitoring tool to constitute our ground truth. Indeed, relying on external tools provided by the testbeds (e.g., Zabbix) could have introduced further relative errors to consider and handle in our experiments. Besides, using those tools could have increased the bandwidth overhead measured by bmon.

[5]https://github.com/di-unipi-socc/FogMon/tree/liscio-2.0/FogMonEye

[6]We consider that a running instance of FogMon2 is stable when (*i*) the FogMon2 overlay organisation into Leaders and Followers does not change for 40 consecutive Leader-Leader communication periods and (*ii*) all direct measurements of links and hardware have been performed at least once in each group and show an error on direct measurements $\leq 25\%$.

(NF) node failures experiments (NF1, NF2 and NF3), testing and assessing the two configurations of FogMon2 against three types of Leader and Follower failures, and
(LF) link failures experiments (LF1 and LF2), testing and assessing the two configurations of FogMon2 against three types of Leader and Follower network degradations.

Node failure experiments target the capability of FogMon2 to self-organise in presence of failures of varying numbers of Leader and Follower nodes, at increasing sizes of the available infrastructure. They consider causing the failure of a number of randomly selected nodes: 50% Leaders and 25% Followers (NF1), all but one Leader and 50% Followers (NF2), and all but one Leader and no Follower (NF3).

On the other hand, link failure experiments target the capability of FogMon2 of adapting to variations in communication links status in terms of available bandwidth and latency. They inject a substantial performance degradation in 5% of the links of the original tree topology built by the *Infrastructure Builder* (LF1) and in 10% of the links of the original tree topology built by the *Infrastructure Builder* (LF2). Also in this case the links are randomly selected.

Experiments NF1–NF3 and LF1–LF2 are repeated for the *Small* and *Large* testbed configurations, and for the default and reactive configurations of FogMon2. Experiments last 30 minutes each and results are averaged across two runs. All collected data is publicly available in [9].

IV. Experiment results

In this section, we illustrate the results of the experiments[7] illustrated in Sect. III by comparing the performance of the *reactive* and *default* configurations of FogMon2 with respect to time to stability (Sect. IV-A), footprint (Sect. IV-B) and relative errors on measurements and estimates (Sect. IV-C). We, finally, briefly contrast the performances of FogMon2 in the *default* and *reactive* configuration (Sect. IV-D).

A. Time to stability

Fig. 3 shows the average time to reach stability across experiments NF1–NF3 (denoted by circles), and LF1–LF2 (denoted by triangles), in the *default* and *reactive* configurations, for both the *Small* and *Large* infrastructure sizes.

Throughout all our experiments, FogMon2 was capable of self-organising in less than 14 minutes, settling on average below 10 minutes. In experiments NF1–NF3, the *default* configuration achieved the highest time to re-organise, almost independently from the infrastructure size. For what concerns the *reactive* configuration, the time to reach stability for the same experiments NF1–NF3 increased as the infrastructure size increased, going from 7 and 10 minutes. Such an increase was naturally due to higher gossiping time across Leaders in larger settings, and to the fact that larger testbeds might incur in longer topology restructurings after node failures.

[7]Experimental data in tabular form, including a *Medium* scale testbed (30 nodes), can be found at https://github.com/di-unipi-socc/FogMon/blob/liscio-2.0/docs/CNERT2022.pdf.

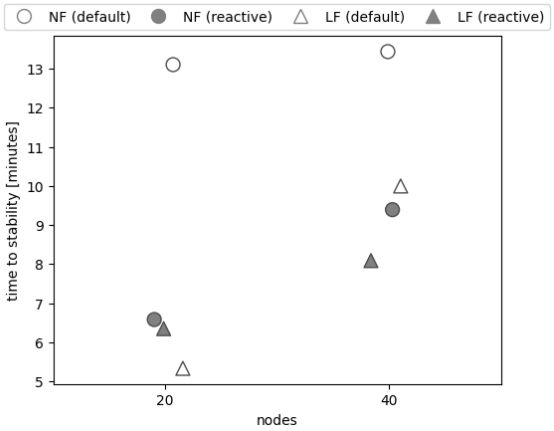


Fig. 3: Time to stability.

Similar considerations hold for LF1–LF2 experiments, where time to stability increased with the infrastructure size. On one hand, the *default* configuration needed between 5 and 10 minutes to reach stability. On the other hand, the *reactive* configuration settled between 6 and 8 minutes. Both configurations showed a similar lower bound (viz. 5 and 6 minutes) due to shorter restructurings in the *Small* testbed.

Overall, across both NF1–NF3 and LF1–LF2, the *reactive* configuration is generally faster to reach stability (with the exception of LF for 20 nodes), showing a time saving of approximately 20% for the *Large* infrastructure and 45% for the *Small* infrastructure.

B. Footprint

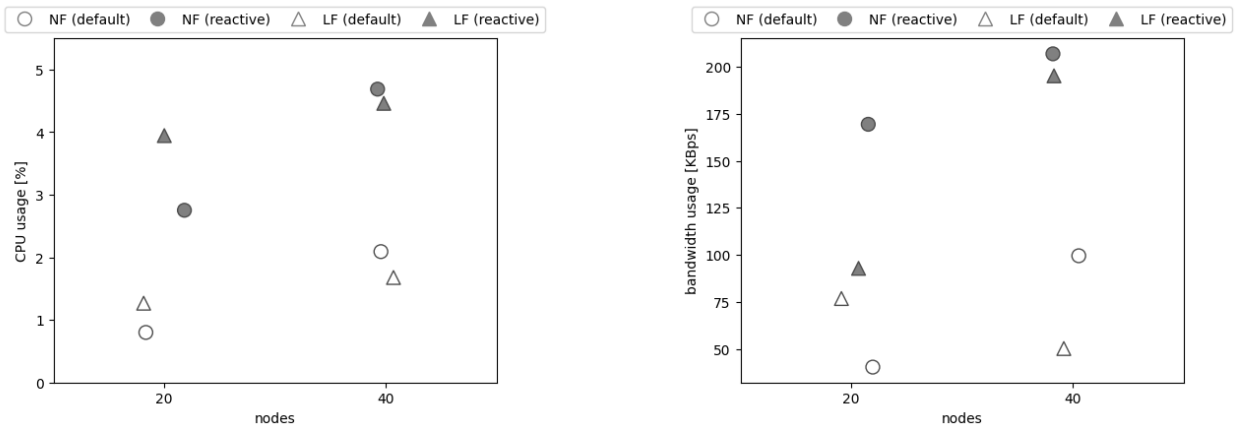
Here we discuss the resource usage of FogMon2 throughout all experiments. We focus here on CPU and bandwidth usage as they have shown significant variations during the experiments. On the contrary, RAM usage settled below 32MB, making FogMon2 suitable also for resource-constrained nodes (e.g., with 256MB of RAM).

Fig. 4 shows the average resource usage of FogMon2 across experiments NF1–NF3 and LF1–LF2, in the *default* and *reactive* configurations, for both the *Small* and *Large* testbeds.

Particularly, Fig. 4a shows the percentage usage due to FogMon2 of one CPU core, and Fig. 4b shows the mean bandwidth usage[8] due to FogMon2 gossiping and bandwidth measurements. Across all experiments, FogMon2 CPU usage required less than 5% of a single CPU core. For both experiment types, node failure and link failure, the CPU usage increased with the size of the infrastructure. Conversely, the *reactive* configuration showed a slightly higher CPU usage (i.e., +2.5% approximately) than the *default* one.

For what concerns bandwidth consumption, FogMon2 required less than 2 Mbps (i.e., 250 KBps) for both upload and download over links that featured an average of 30 Mbps

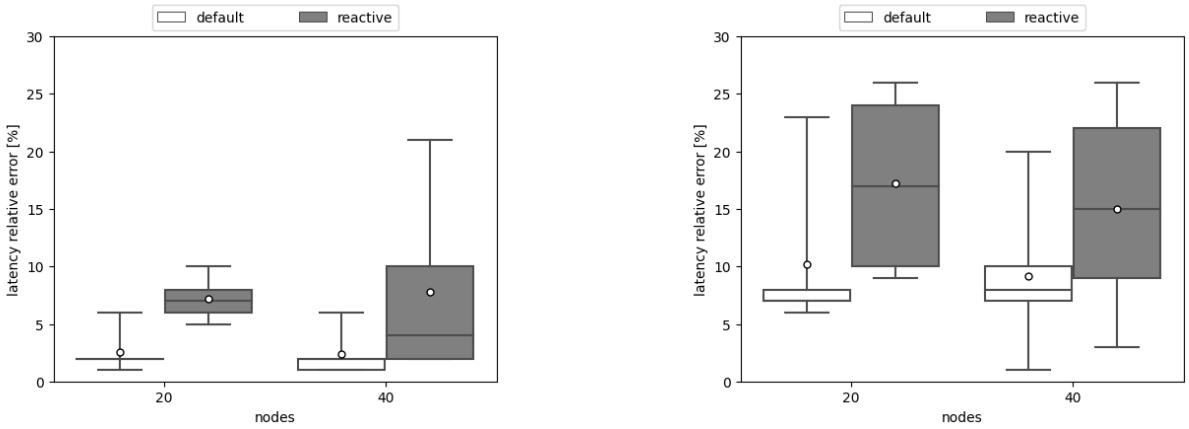
[8]For the sake of conciseness, as the results for the upload and download bandwidth were almost identical, we only illustrate here the results obtained for the upload bandwidth.



(a) CPU usage (1 core)

(b) Bandwidth usage

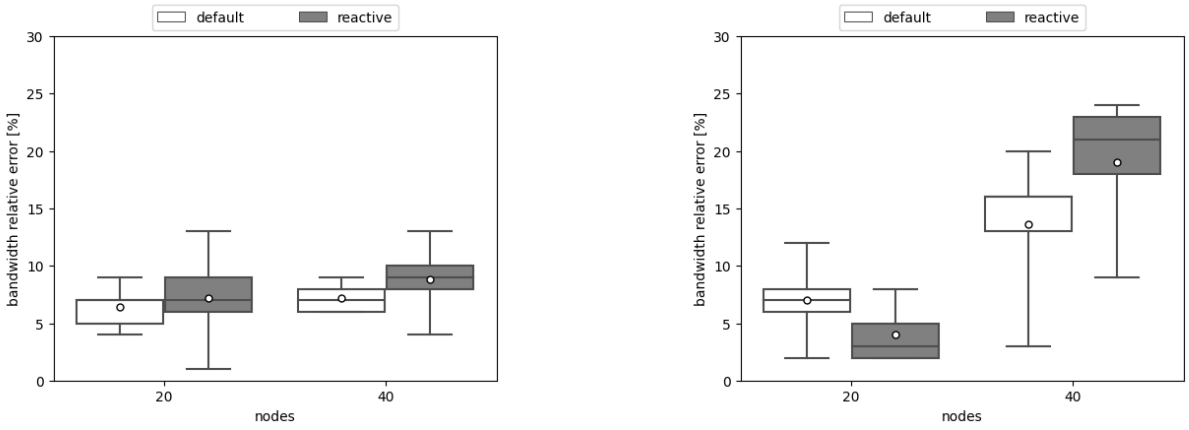
Fig. 4: Resource footprint of FogMon2.



(a) Intra-group latency

(b) Inter-group latency

Fig. 5: Relative errors of FogMon2 on latency (aggregated for NF and LF experiments).



(a) Intra-group bandwidth

(b) Inter-group bandwidth

Fig. 6: Relative errors of FogMon2 on bandwidth (aggregated for NF and LF experiments).

bandwidth. The increase in the infrastructure size translated in an increase of the bandwidth usage in most cases (i.e., NF1–NF3, and *reactive* LF1–LF2 in Fig. 4b). Overall, the *reactive* configuration required on average 0.8Mbps (100KBps) more in comparison with the *default* configuration.

C. Relative errors

In the following, we briefly discuss the relative error on both actual measurements and estimates performed by FogMon2 on latency and bandwidth, as illustrated in Fig. 5 and 6, respectively. In all figures, we use box-plots to represent the relative error across all experiments (NF1–NF3 and LF1–LF2) for the *default* and *reactive* configurations. Boxes indicate the maximum and minimum values (upper and lower whisks, respectively), and the 25^{th}, 50^{th} and 75^{th} percentiles (indicated by the box). A white circle denotes the average value.

a) Latency: Fig. 5 shows that, across our experiments, the errors on latency were less than 26%. Independently from the infrastructure size, intra-group direct measurements (Fig. 5a) showed an average relative error between 2.4% and 7.8%, and inter-group estimates (Fig. 5b) between 9.2% and 17.2%. The *reactive* configuration achieved worse results in comparison with the *default* configuration. This is explained by the higher bandwidth usage of the *reactive* configuration (see also Sect. IV-B) that affects latency measurements. In turn, being based on those measurements, estimates showed a higher relative error in the *reactive* configuration.

b) Bandwidth: Fig. 6 shows that, across our experiments, the errors on bandwidth were always less than 24%. Intra-group direct measurements (Fig. 6a) showed an average relative error between 6.4% and 8.8%, and inter-group estimates (Fig. 6b) between 4% and 19%. The *reactive* configuration showed higher variability in the relative error on bandwidth measurements, compared with the *default* configuration. This is due to the higher bandwidth probing frequency that possibly led to interference in the coordination of bandwidth measurements from different nodes. Intra-group measurements (Fig. 6a) showed worse results in the *Large* testbed, due to the higher number of monitored end-to-end links w.r.t. the *Small* testbed (i.e., 1600 vs. 400).

D. Default versus Reactive

To sum up, both the *default* and *reactive* configurations of FogMon2 were capable of handling correctly the infrastructure variations triggered by our experiments, maintaining a correct functioning through suitable overlay restructurings. As expected, direct intra-group measurements feature lower relative errors with respect to inter-group estimates, in both configurations.

In comparison with the *default* configuration, the *reactive* configuration typically ensures faster time to stability, i.e., faster identification of changes in the monitored infrastructure (e.g., node crash, link QoS degradation). This aspect can be important when monitored data need to be used to decide on the management of quasi-realtime or mission-critical application services. Naturally, such an improvement in FogMon2's reactiveness comes at the cost of a (slightly) higher resource usage and to an increase in the relative error on measurements and estimates that mainly derives from network congestion due to more frequent measurements.

V. Concluding Remarks

In this article, we described and assessed an improved new version of our lightweight decentralised monitoring tool for Fog computing infrastructures, FogMon2.

In our experiments, FogMon2 was capable of maintaining its correct functioning via topology restructurings. It shows a contained footprint. RAM usage is always bound by 30MB, which makes it usable on resource-constrained edge devices (e.g., with 512MB of RAM), and CPU usage is bound by 5% at worst. Similarly, for bandwidth usage, the default configuration settles around 70 KBps, while the reactive one around 170 KBps. This confirms FogMon2 is lightweight and avoids overloading situations due to monitoring. Such performance suggests that FogMon2 is suitable for monitoring Fog resources, featuring a good compromise between intrusiveness and measurements accuracy.

The experiments carried out in LiSCIo brought FogMon from TRL4 to TRL5. Last, but not least, FogMon2 has been extended with suitable tooling and a GUI, FogMonEye, which will allow us to assess new future releases of our tool, e.g., across different opportunistic-like network topologies.

Finally, we intend to integrate FogMon2 with our continuous reasoning methodologies for application management [10] and with an existing orchestration platform, so to prototype an autonomic QoS-aware tool for managing next-gen multiservice applications on top of a Fog infrastructure.

References

[1] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, and V. Stankovski, "Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review," *J. Syst. Softw.*, vol. 136, pp. 19–38, 2018.

[2] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Comput. Surv.*, vol. 53, no. 3, pp. 1–35, 2020.

[3] S. Forti, M. Gaglianese, and A. Brogi, "Lightweight self-organising distributed monitoring of Fog infrastructures," *Future Gener. Comput. Syst.*, vol. 114, pp. 605–618, 2021.

[4] Á. Brandón, M. S. Pérez, J. Montes, and A. Sanchez, "Fmone: A flexible monitoring solution at the edge," *Wireless Communications and Mobile Computing*, 2018.

[5] A. Souza, N. Cacho, A. Noor, P. P. Jayaraman, A. Romanovsky, and R. Ranjan, "Osmotic monitoring of microservices between the edge and cloud," in *HPCC/SmartCity/DSS 2018*, 2018, pp. 758–765.

[6] A. Brogi, S. Forti, and M. Gaglianese, "Measuring the fog, gently," in *International Conference on Service-Oriented Computing, (ICSOC)*, ser. LNCS, vol. 11895. Springer, 2019, pp. 523–538.

[7] E. Schubert and P. J. Rousseeuw, "Faster k-medoids clustering: Improving the pam, clara, and CLARANS algorithms," in *SISAP*, 2019.

[8] M. U. and B. S., "Performance evaluation of some clustering algorithms and validity indices," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 12, p. 1650 – 1654, 2002.

[9] M. Gaglianese, S. Forti, F. Buti, F. Paganelli, and A. Brogi, "Lightweight Self-adaptive Cloud-IoT Monitoring across Fed4FIRE+ Testbeds (LiSCIo) – Dataset," http://doi.org/10.5281/zenodo.4682986, 2021.

[10] S. Forti, G. Bisicchia, and A. Brogi, "Declarative Continuous Reasoning in the Cloud-IoT Continuum," *Journal of Logic and Computation*, 2022.