MonEx: An Integrated Experiment Monitoring Framework Standing on Off-The-Shelf Components

Abdulqawi Saif, Alexandre Merlin, Lucas Nussbaum, Ye-Qiong Song

MonEx: An Integrated Experiment Monitoring Framework Standing on Off-The-Shelf Components

Abdulqawi Saif
Université de Lorraine
Qwant Entreprise
F-54000 Nancy, France
abdulqawi.saif@univ-lorraine.fr

Alexandre Merlin
Université de Lorraine, CNRS, Inria, LORIA
F-54000 Nancy, France
alexandre.merlin@inria.fr

Lucas Nussbaum
Université de Lorraine, CNRS, Inria, LORIA
F-54000 Nancy, France
lucas.nussbaum@loria.fr

Ye-Qiong Song
Université de Lorraine, CNRS, Inria, LORIA
F-54000 Nancy, France
ye-qiong.song@loria.fr

ABSTRACT

Most computer experiments include a phase where metrics are gathered from and about various kinds of resources. This phase is often done via manual, non-reproducible and error-prone steps. Infrastructure monitoring tools facilitate collecting experiments' data to some extent. However, there is no conventional way for doing so, and there is still much work to be done (e.g. capturing user experiments) to leverage the monitoring activity for monitoring experiments. To overcome those challenges, we define the requirements of experiments monitoring, clarifying *Experiment Monitoring Frameworks*' scope and mainly focusing on reusability of experiments' data, and portability of experiments' metrics. We then propose *MonEx EMF* that satisfies those requirements. *MonEx* is built on top of infrastructure monitoring solutions and supports various monitoring approaches. It fully integrates into the experiment workflow by encompassing all steps from data acquisition to producing publishable figures. Hence, *MonEx* represents a first step towards unifying methods of collecting experiments' data.

CCS CONCEPTS

• **Networks** → *Network experimentation*;

KEYWORDS

testbed frameworks, experimentation, experiment monitoring

ACM Reference Format:
Abdulqawi Saif, Alexandre Merlin, Lucas Nussbaum, and Ye-Qiong Song. 2018. MonEx: An Integrated Experiment Monitoring Framework Standing on Off-The-Shelf Components. In *P-RECS'18: First International Workshop on Practical Reproducible Evaluation of Computer Systems, June 11, 2018, Tempe, AZ, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3214239.3214240

P-RECS'18, June 11, 2018, Tempe, AZ, USA

ACM ISBN 978-1-4503-5861-3/18/06…$15.00
https://doi.org/10.1145/3214239.3214240

1 INTRODUCTION

Most computer science experiments involve a phase of data acquisition, during which metrics are collected about the system under study. This phase has a central role in the experimental process. First, it is the conclusion of the experiment per-se, after the steps of experiment design, setup, and execution. But the collected raw data is also the starting point for the phase of data analysis that should lead to trustworthy, reproducible, and publishable results. One would expect data acquisition to be performed with well-designed solutions, that fully integrate in the experiment workflow, maximize support for reproducibility of experiments, and limit the risk of user errors. However, in practice, experimenters often resort to ad hoc and manual solutions, such as writing *dumps* or logs, gathering them manually, and parsing them using custom scripts.

Many monitoring tools such as *Ganglia* [7] are in use for monitoring various platforms' infrastructure (e.g. clouds, testbeds). They provide an overview of resources status and usage, raising alerts when things go wrong. Beside their goal of permanently providing infrastructure monitoring, These tools can be used with additional effort for monitoring experiments. But no framework is found for addressing monitoring experiments, and encompassing all experimentation steps that vary from collecting experiments data to creating publishable figures. Thus, our main concern is to examine the idea of building *Experiment Monitoring Frameworks (EMFs)* on top of the state of the art of infrastructure monitoring tools.

The first contribution of this paper is defining a list of requirements to be satisfied by *EMFs*. We link those requirements with the current state of the art in a literature review, showing that no solution experiences a full coverage. We then implement *MonEx* (***Mon****itoring* ***Ex****periments*) *EMF*. *MonEx* covers all defined requirements, and nicely inserts into the experiment workflow.

This paper is organized as follows. In Section 2, we discuss the specific requirements and challenges for monitoring experiments. We then analyze the positioning of related work in Section 3. *MonEx* is described in Section 4, before being featured in use case experiments (Section 5). Finally, we conclude with Section 6.

2 REQUIREMENTS AND CHALLENGES

An ideal *Experiment Monitoring Framework* (*EMF*) should meet a number of requirements, which are detailed below.

Table 1: Identified requirements for experiment monitoring (Section 2) vs related work (Section 3)

	Infrastructure monitoring tools e.g. Munin	Testbed-provided measurement services	Vendetta[12]	OML[15]
Experiment-focused	-	-	-	-
Independent of experiments	+	+	-	-
Independent of testbeds services	+	+	-	+
Scalability	+	+	-	+
Low impact	-	+	-	-
Easy deployment	+	-	+	+
Controllable	-	-	+	+
Real-time monitoring	+	+	-	-
Producing publication-quality figures	-	-	-	-
Archival of data	+	+	-	+

Experiment-focused. The notion of *Experiment* should be central in the *EMF*. It should keep track of experiments' name or identifier, start time, end time, and list of associated metrics. This should maintain an overview of the different experiments performed by the same or different users.

Independent of experiments. An *EMF* should support a wide range of experiments, regardless of the number of metrics, the frequency of measurements, or the software or services being monitored. Furthermore, it should not be necessary to alter the system under test for it to be monitored by such a tool. This helps to reproduce the experiment on other testbeds even if the *EMF* is absent.

Independent of testbed services and experiment management frameworks. Building the monitoring facility into the core testbed services or management framework, as an all-in-one solution, has some advantages. However, an *EMF* should ideally maintain a high level of independence from such services to facilitate porting experiments to other testbeds, or monitoring experiments on federations of heterogeneous testbeds.

Scalability. An *EMF* should scale to a large number of monitored resources, to a large number of metrics, and to high-frequency of measurements, in order to allow understanding fine-grained phenomena (at the millisecond scale), or phenomena that only occur with hundreds or thousands of nodes.

Low impact. The *EMF* should have a low impact on the resources involved in the experiment in order to avoid the *observer effect* (the addition of monitoring causing significant changes to the experiment's results).

Easy deployment. An *EMF* should not depend on complex or specific testbed infrastructure. It should be easy to deploy over any networking or distributed testbeds without tedious configuration.

Controllable. An *EMF* should be flexible. Users should have the choice to enable or disable the monitoring of their experiments at any time, and to select metrics, e.g. in order to limit or evaluate the impact of the *EMF* on the experiment.

Real-time monitoring. The *EMF* should provide real-time feedback during the experiment execution, to allow the early detection of issues in long-running experiments.

Producing publication-quality figures. The *EMF* should integrate the final step of the experiment life-cycle, that is turning results into publishable material, with minimal additional effort.

Archival of data. Saving and exporting the experimental metrics of a given experiment is important to allow for future analysis of the data. It is also a basis for allowing distribution in an open format to enable others to repeat the analysis.

3 LITERATURE REVIEW

To distinguish from previous works, we describe the state of the art of infrastructure monitoring tools, testbed measurement services, and instrumentation frameworks.

Infrastructure monitoring tools. Infrastructure monitoring is a frequent need for system administrators, to track resources utilization, errors, and get alerted in case of problems. Many tools exist, from the ancestor MRTG [9], to *Munin*, *Nagios*, *Ganglia* [7], *Zabbix* [10], or *Cacti*. They differ in terms of design choices such as protocols used to query resources, use of remote agents to collect and export metrics, or the way of collecting and storing data (*push* vs *pull*). However, they all target the monitoring of long-term variations of metrics, and thus are designed for relatively long intervals between measurements (typically 5 to 10 minutes). They don't scale well to shorter intervals, which makes them unsuitable for monitoring fine-grained phenomas during experiments. Additionally, most of them rely on *RRDtool* to store metrics and generate figures, which is a suitable tool for the infrastructure monitoring use case, but not well suited at all for generating publication-quality figures.

More recently, with the emergence of elasticity and cloud infrastructures, more modern infrastructure monitoring tools were designed, such as Google's *Borgmon*, *gnocchi*, *Prometheus* or *InfluxDB*. *Prometheus* and *SNMP* are used in [2] to build an agent-less monitoring system. In *Sec.* 4 we will describe our own effort to base our framework on *Prometheus* and InfluxDB.

Testbed-provided measurement services. Some testbeds provide services to expose some metrics that would otherwise not be available to experimenters. For example, the *Grid'5000* testbed [1] provides *Kwapi* [4] for network traffic and power measurements, collected respectively at the network equipment and at the power distribution unit. *PlanetLab* [3] used *COMon* to expose statistical information about the testbed nodes and the reserved slices. In general, these services are limited to very specific metrics. Thus, experimenters have no permission to add their own metrics or to

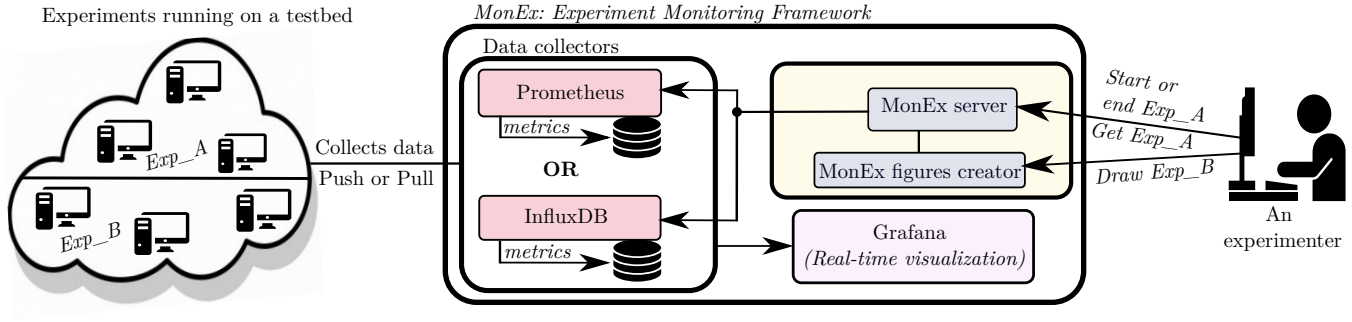

Figure 1: Overall design of the *MonEx EMF*

otherwise customize these services for their experiments. Those services should rather be considered as potential additional sources of information for an experiment monitoring framework.

Instrumentation frameworks. There are very few attempts at providing frameworks that address the specific needs of experimentation. One of these attempts is *Vendetta*[12] which is a simple monitoring and management tool for distributed testbeds. It runs an agent code on every node to be monitored to parse the experiment events before sending the results to the central sever which does the visualization mission. *Vendetta* has no mechanism to implement the starting and the ending time of experiments, so the researcher must manually track the experimental timing in order to extract the collected metrics. In addition, the client agent can restart the monitoring server (running on another node) in case of lack of response, which could be problematic in some cases.

Another solution is *OML* [15, 16] (*ORBIT Measurement Library*), which is closely related to the *OMF* [11] testbed management framework. With *OML*, the experiment components stream their measurements towards an *OML* server. The server creates a SQL database to store the metrics of each experiment. The process of using *OML* is experiment-dependent: several steps are required to modify the experimental code to define the measurement points. In addition, the *OML* server does not provide real-time monitoring – the only way to access the metrics is to query the experiment database. Overall, *OML* has seen rather low adoption, even if some testbeds like *Fibre* [13], *IOT-Lab*, or *NITOS* [5] support it. Its development seems to have been stalled (last changes on GitHub in 2015).

In Tab. 1, we compare the tools presented here with the requirements discussed in Sec. 2. As can be seen, the existing tools fail to match all requirements, which triggered the design of our own solution, described in the next section.

4 *MONEX* DESIGN

This section introduces *MonEx*, our integrated Experiment Monitoring Framework (*Fig. 1*). Inspired by the *Popper* convention [6], we reuse some off-the-shelf monitoring technologies that fit into *MonEx* design rather than making new ones, and then build on top of them to adjust to the specific requirements of experiment monitoring. Thus, *Prometheus* and *InfluxDB* are used as *data collectors* while *Grafana* is used for *real-time visualization*. But those off-the-shelf components are complemented with custom-built components. First, *MonEx server* brings the experiment process to the monitoring solution, by enabling the experimenter to specify the experiments' start and end time in order to link metrics to specific experiments (allowing the extraction of an experiment's metrics, or to refer to a specific experiment for analysis or comparison purposes). Second, *MonEx figures-creator* makes it possible to automatically extract metrics for a specific experiment, and create publishable figures.

MonEx is designed as a command line tool. Experimenters directly interact with the *MonEx-* server or the figure-creator. After setting up an experiment, two calls to the *MonEx server* are issued at the beginning and the end of that experiment, to allow specifying the experiment time boundaries. The server also deals with the experimenters requests to query their experiments. *Prometheus* and *InfluxDB* are used to retrieve the experiment metrics from the execution environment, representing the main data source of *MonEx*. Hence, experiments can use an appropriate monitoring technique (e.g. *agent or agent-less monitoring, and pull or push monitoring*). Furthermore, *Grafana* is used to visualize the experiment metrics at runtime by connecting to the data collectors as a consumer. At last, when the experiment is accomplished, *MonEx server* is able to produce an output file containing the target metrics of a given experiment. Eventually, *MonEx figures-creator* either exploits that file (e.g. in case of running in another environment), or connects directly to the *MonEx server* in order to generate publishable figures.

The components of *MonEx* are described in detail in the following sections.

4.1 *MonEx* server

MonEx HTTP server is built to handle the time boundaries of experiments as well as manipulating their metrics. It exposes different interfaces to receive notifications about the start and the end time of experiments, to query the experiments metrics, and to remove experiments from its list. Each experiment sends at least two *HTTP* requests: *start_xp* to indicate its intention to use *MonEx*, bringing also a name for this experiment to be distinguished by, *end_xp* to notify the server about the experiment termination time. The requests could be integrated inside the experiment code to be more dynamic when declaring the time boundaries, especially for short-length experiments. *MonEx server* is also used to query the experiment metrics using *get_xp* request. This request asks the *MonEx server* to expose the desired metrics into a *CSV* file. For example, to export a

metric named *mymetric* of the experiment *myexp* into a *CSV* file, the following command could be used:

```
curl "htttp://MonExServer:5000/exp/myexp"
-X GET -d '{"metric":"mymetric"}' > file.csv
```

MonEx server requires a configuration file which describes the data collector in use. This helps to send specific commands during communication. The server could also connect to several instances of the data collectors simultaneously, enabling experimenters to interact with multiple data collectors (e.g. experiments running in different physical sites of the same testbed where each site provides its own data collector).

MonEx server tasks can be differently achieved by adding a control metric into *Prometheus*. However, this alternative has various limitations. Firstly, it will not scale as every experiment will require an independent instance of *Prometheus* to decode the added control metric. Thus, it will be difficult to have a service for monitoring experiments made available to all testbed users. Secondly, it limits the use of the underlaying monitoring system to only *Prometheus*, ignoring the experiments that use other collectors. Thirdly, even if *Prometheus* has a lot of ready-to-use exporters, modifying each of them by adding control-metrics will prevent experimenters from using them directly. Taking these limitations into account, we choose to keep track of experiments via a dedicated server.

4.2 Experiments data collectors

MonEx supports the use of either *Prometheus* or *InfluxDB* in order to cover all monitoring techniques. *Prometheus* is a monitoring system with alerting and notification services and a powerful querying language which allows creating compound metrics from existing ones. It is optimized to pull numerical metrics into a central server, but not to scale horizontally or to support non numerical metrics as *InfluxDB* does. *InfluxDB* is a chronological time series database for storing experimental metrics with a timestamp resolution that scales from milliseconds to nanoseconds. In *MonEx*, both could be mutually or simultaneously used regarding the experiment need. Indeed, they provide similar services regardless of their differences.

Although *Prometheus* is the default data collector in *MonEx*, its differences with *InfluxDB* favorite this latter for specific use cases. Firstly, *InfluxDB* fits better for the experiments that send their metrics in variable-time intervals since *Prometheus* still needs to pull the data regarding his scraping interval (even if that makes no sense for the experiment). For example, pulling the metrics every second is not significant for the experiment that generates its data at random time intervals, so pushing them into *InfluxDB* whenever the experiment has new data is more preferable. Secondly, as it follows the *pull-based* approach, *Prometheus* is not able to collect data from the experiments with high frequency measurements since its scraping interval does not go beyond one second (it is also true if *Prometheus-Pushgateway* is used along with *Prometheus*). Thus, using *InfluxDB*, which supports pushing data at high scale, is a robust solution to prevent any data loss during such experiments.

4.3 *MonEx* figures-creator

This component is essential to exploit the monitoring results for creating publishable figures. It is a tool that deals with the export of an experiment's data from MonEx into a format (CSV) that is widely supported by tools typically used to prepare figures (*R*'s *ggplot*, *gnuplot*, *matplotlib*, *pgfplot*, etc.). It also includes direct support for generating figures using R, covering a wide range of standard figures (e.g. X-Y figures, stack figures, multiple-Y figures, ..., etc).

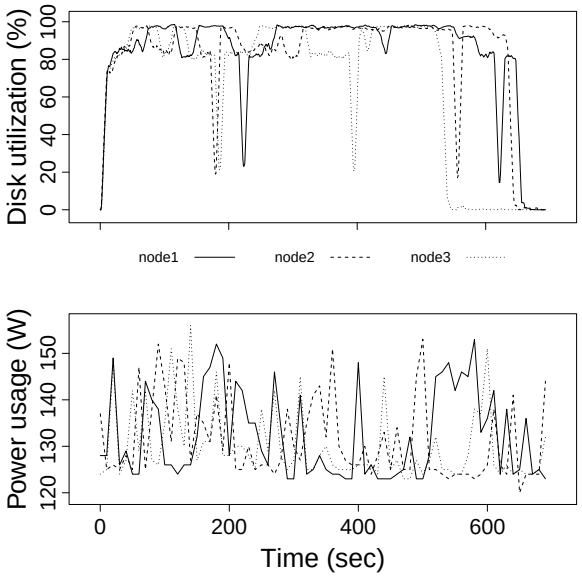

Figure 2: Disk utilization affecting the power consumption.

4.4 Real-time visualization

MonEx uses *Grafana* for real-time visualization using a modern web-based interface. *Grafana*, which works on time series, consumes the available metrics collected by *Prometheus* and *InfluxDB*. However, the experiments with high frequency measurements trigger a trade-off with the real-time visualization as they might impact the experiment resources by producing a massive volume of data. Thus, such experiments should be configured either to push its metrics entirely at the end, making this service totally unusable, or to push them over periodic chunks to still benefiting from this service with a reduced precision.

5 USE CASE EXPERIMENTS

This section highlights how *MonEx* covers all the requirements listed in Section 2 by describing three experiments. Each experiment mainly stresses partial requirements while all of them are performed on a homogeneous cluster of the *Grid'5000* testbed.

5.1 Disk & Power Usage of a *MongoDB* cluster

This experiment evaluates the *disk utilization* and the *power usage* of a three-shards cluster of *MongoDB*, while performing an indexing workload over a 80 GB of data.

Prometheus SNMP- & node- exporter are used to tackle the cluster power consumption and the disk utilization, respectively. On the one hand, *Prometheus SNMP Exporter* is used on the power distribution units (PDUs) to obtain the power per outlet. It is installed on one machine, but it queries all concerned machines thanks to

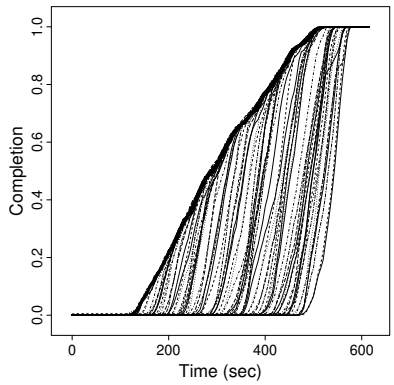

Figure 3: Torrent completion results (one line per peer).

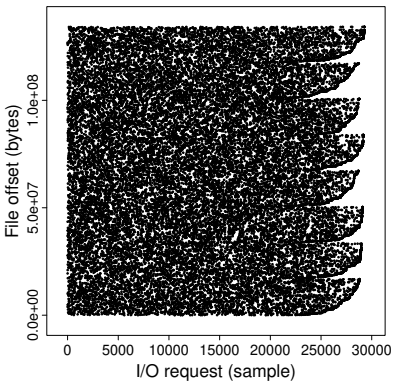

Figure 4: I/O pattern of *FIO*'s *randread* over a 140 MB file.

adding their PDU addresses into its configuration file. Using *agentless monitoring* reduces the impact on the monitored environment. One the other hand, *Prometheus node exporter* is installed on each machine to report the *disk* usage per machine.

Two instructions are only added into the experiment script in order to notify *MonEx* about its start and the end time. We obtain our target metrics by sending a customized *get_xp* request to the *MonEx server*. *MonEx figures-creator* is then used to generate a publishable figure that contains the target metrics. *Fig. 2* contains three curves that represent the disk utilization and the power usage of the deployed cluster.

5.2 Many-nodes Bittorrent download

We revisit the torrent expriment covered in [8] using *MonEx*. Monitoring the torrent completion of a given file is the main metric of this experiment. A seeder with a 500 MB file is created and multiple peers seek to download the target file. A mesh topology is used for connecting the seeder/peers, while *Transmission* is used as a torrent client for the peers.

The network and the peers are emulated by *Distem* [14], so the experiment runs independently from the testbed topology. The seeder bandwidth is limited to 5 KB/s while this of peers is limited to 30 KB/s. Each peer resides on a virtual node, and all nodes are increasingly connected with a constraint that a new peer is entering the network every 4 seconds until the number of peers reaches its maximum (100 peers).

The experiment begins when the seeder shares the target file by notifying the tracker. It terminates when all peers have that file data. To obtain the completion of the target file, we query the *Transmission* API using our own metrics exporter (about 10 lines of *Python*). The exporter is instantiated to run on each virtual node, while *Prometheus* pulls periodically their data. *MonEx* has a minimal impact on the experiment resources as there is another dedicated *VLAN* in use for the monitoring traffic. This experiment shows how *MonEx* is scalable, as *Prometheus* pulls the experiment metrics from about a hundred of nodes without any overflow.

MonEx is then used either to produce a *CSV* file containing the experiment metrics selected by the experimenter, or to obtain directly a ready-to-publish graph, as shown in *Fig. 3*.

5.3 Experiment with time-independent metric

This experiment evaluates how a data file is accessed during a workload execution. We use the *Fio* benchmark to generate an I/O access pattern over a given file. In parallel, we create an *extended Berkley Packet Filter* (*eBPF*) tool to uncover this access pattern. The target metric is the file offsets. If the access is random, we are expecting to see a shapeless view of file offsets versus the sequences of the I/O requests.

This experiment is challenging in both scalability and controllability. Firstly, a pull-based monitoring cannot be used since the experiment has high frequency measurements. Hence, a scraping interval even of one second might not catch all events (data exposed to be lost). Secondly, the target metric does not rely on the timestamps, but rather on the order of I/O requests accessing the file. Hence, every I/O request is significant for understanding the overall access pattern. For these two reasons, we use *InfluxDB* rather than *Prometheus*. We locally collect the massive I/O requests before pushing them at once to *InfluxDB* at the end of experiment. *Figure 4* shows I/O patterns of a random read workload. The file offsets are totally shapeless regarding the sequences of the issued I/O requests.

6 CONCLUSIONS

In this paper, we firstly defined the needed requirements to build an *experiment monitoring framework* (*EMF*). We then leveraged these requirements and some recent infrastructure monitoring solutions to introduce the *MonEx EMF*. Through use cases, we showed how *MonEx* reduces the experimenters' effort by encompassing all the steps from collecting metrics to producing publication quality figures, leaving no places for manual and ad hoc steps that were used to be performed in practice.

MonEx has two impacts on the way we perform experiments. Firstly, it pushes towards the repeatability of experiments' analysis and metrics comparison. That is thanks to its abilities to separate the phase of collecting metrics from experiments, and to archive them per experiment. Secondly, as *MonEx* puts the experiment at the center of the monitoring workflow, focusing on how the experiment results are obtained rather than where the experiment runs, we are a step closer to better experiment portability and reproducibility.

REFERENCES

[1] Daniel Balouek et al. 2013. Adding Virtualization Capabilities to the Grid'5000 Testbed. In *Cloud Computing and Services Science*. Communications in Computer and Information Science, Vol. 367. 3–20. https://doi.org/10.1007/978-3-319-04519-1_1

[2] Morgan Brattstrom and Patricia Morreale. 2017. Scalable Agentless Cloud Network Monitoring. In *Cyber Security and Cloud Computing (CSCloud)*.

[3] Brent Chun et al. 2003. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review* (2003).

[4] Florentin Clouet et al. 2015. A unified monitoring framework for energy consumption and network traffic. In *TRIDENTCOM*.

[5] Dimitris Giatsios, Apostolos Apostolaras, Thanasis Korakis, and Leandros Tassiulas. 2013. Methodology and tools for measurements on wireless testbeds: The nitos approach. In *Measurement Methodology and Tools*. Springer, 61–80.

[6] Ivo Jimenez, Michael Sevilla, et al. 2017. The Popper Convention: Making Reproducible Systems Evaluation Practical. In *Parallel and Distributed Processing Symposium Workshops (IPDPSW)*.

[7] Matthew L Massie et al. 2004. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Comput.* (2004).

[8] Lucas Nussbaum and Olivier Richard. 2008. Lightweight emulation to study peer-to-peer systems. *Concurrency and Computation: Practice and Experience* 20, 6 (2008), 735–749.

[9] Tobias Oetiker and Dave Rand. 1998. MRTG: The Multi Router Traffic Grapher.. In *LISA*, Vol. 98. 141–148.

[10] Rihards Olups. 2010. *Zabbix 1.8 network monitoring*. Packt Publishing Ltd.

[11] Thierry Rakotoarivelo, Maximilian Ott, Guillaume Jourjon, and Ivan Seskar. 2010. OMF: a control and management framework for networking testbeds. *ACM SIGOPS Operating Systems Review* 43, 4 (2010), 54–59.

[12] Olof Rensfelt, Lars-Ake Larzon, and Sven Westergren. 2007. Vendetta – a tool for flexible monitoring and management of distributed testbeds. In *TridentCom*.

[13] Tiago Salmito, Leandro Ciuffo, Iara Machado, et al. 2014. FIBRE-an international testbed for future internet experimentation. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.

[14] Luc Sarzyniec, Tomasz Buchert, Emmanuel Jeanvoine, and Lucas Nussbaum. [n. d.]. Design and Evaluation of a Virtual Experimental Environment for Distributed Systems. In *PDP2013 - 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing*.

[15] Manpreet Singh, Maximilian Ott, Ivan Seskar, and Pandurang Kamat. 2005. ORBIT Measurements framework and library (OML): motivations, implementation and features. In *Tridentcom*.

[16] https://github.com/mytestbed/oml. [n. d.]. ORBIT Measurement Library.