Achieving Reproducible Network Environments with INSALATA

Nadine Herold, Matthias Wachs, Marko Dorfhuber, Christoph Rudolf, Stefan Liebald(✉), and Georg Carle

Department of Informatics, Chair of Network Architectures and Services, Technical University of Munich (TUM), Munich, Germany
{herold,wachs,liebald,carle}@net.in.tum.de,
{marko.dorfhuber,christoph.rudolf}@tum.de

Abstract. Analyzing network environments for security flaws and assessing new service and infrastructure configurations in general are dangerous and error-prone when done in operational networks. Therefore, *cloning* such networks into a dedicated test environment is beneficial for comprehensive testing and analysis without impacting the operational network. To automate this reproduction of a network environment in a physical or virtualized testbed, several key features are required: (a) a suitable network model to describe network environments, (b) an automated acquisition process to instantiate this model for the respective network environment, and (c) an automated setup process to deploy the instance to the testbed.

With this work, we present INSALATA, an automated and extensible framework to reproduce physical or virtualized network environments in network testbeds. INSALATA employs a modular approach for data acquisition and deployment, resolves interdependencies in the setup process, and supports just-in-time reproduction of network environments. INSALATA is open source and available on Github. To highlight its applicability, we present a real world case study utilizing INSALATA.

Keywords: Infrastructure Information Collection · Automated Testbed Setup and Configuration · Testbed Management

1 Introduction

The increasing number of attack vectors and the growing complexity of attacks on computer networks force operators to continuously assess and improve their networks, services, and configurations. Analyzing, testing, and deploying new security features and configuration improvements is time-consuming, challenging, and error-prone. The same holds for general software upgrades or network infrastructure changes. Performing this on an operational network is often not suitable, as service continuity has to be ensured and outages cannot be tolerated.

Therefore, reproducing a network environment into a self-contained test environment is beneficial as the operational network is not influenced. Testing and

D. Tuncer et al. (Eds.): AIMS 2017, LNCS 10356, pp. 30–44, 2017.
DOI: 10.1007/978-3-319-60774-0_3

analyzing different options to improve the network and its security can be evaluated and tested sufficiently before deployment. With large and complex network environments, reproducing such network environments cannot be done manually as information about the environment and its elements may be unknown, incomplete, or not available in a formal description. Hence, an automated process to reproduce network environments in a physical or virtualized testbed is required.

In this work, we present *INSALATA*, the *IT NetworkS AnaLysis And deploymenT Application*. INSALATA enables network operators and researchers to automate reproduction of arbitrary network environments in physical or virtualized testbeds. To represent network environments, we provide a network model comprising layer-2 network segments, IP networks, connectivity information (routing and firewalling), network nodes (routers, hosts), network services (DNS, DHCP), and host information (network interfaces, memory, disks, operating system). INSALATA can analyze network environments to obtain a formal *description* of the network to track the state continuously or in discrete intervals. Here, INSALATA uses *information fusing* to provide a comprehensive view on the network by aggregating information from multiple *collector modules*. Using descriptions decouples analysis and deployment and enables re-using, archiving, and distributing these descriptions. INSALATA can instantiate descriptions on physical or virtualized testbeds employing a PDDL planner to structure the setup process and resolve inter-dependencies between setup steps. To minimize setup steps and reuse existing testbed setups, we support *incremental setups* by determining the delta between current and target testbed state.

The key contributions of our work are (a) INSALATA, a fully automated, modular, and extensible framework to reproduce network environments on testbeds, (b) the open source implementation of INSALATA available on GitHub, and (c) a case study showing INSALATA's applicability to real world scenarios using exemplary module implementations.

The remainder of this paper is structured as follows: First, we describe our goals and requirements for INSALATA in Sect. 2. Afterwards, we analyze if related work can fulfill these in Sect. 3. In Sect. 4, we present INSALATA's design and introduce its components. Next, we present the main components of INSALATA in detail, in particular the underlying information model in Sect. 5, the *Collector Component* in Sect. 6, and the *Deployment Component* in Sect. 7. In Sect. 8, we summarize important implementation details. In Sect. 9, we present a case study to show the applicability of our proposed system. Finally, we give a conclusion and present future work in Sect. 10.

2 Goals and Requirements

The overall goal is to reproduce arbitrary network environments into physical or virtualized testbeds. Therefore, we need (a) a suitable information model reflecting required information, (b) an automated information acquisition process, and (c) an automated deployment process.

(a) Information Model: The information model abstracts from the network environment. The goal is to reflect network environments up to application layer of the TCP/IP reference model. The information model has to be extensible to allow to add use case specific services and additional information elements. Therefore, we require the following information to be present: (a) basic network nodes, like hosts and routers, (b) networks on layer 2 and 3, including appropriate addressing schemes, (c) connectivity information like routing and firewalling, (d) basic network services like DNS and DHCP, and (e) host information, including network interfaces, disks, memory, CPUs, or operating system.

(b) Information Acquisition: The goal is to provide information acquisition that supports different types of information collection techniques, supports continuous monitoring of the network environment, and is fully automated. We identified that the following information collection techniques, differing in terms of intrusiveness and quality of information they provide, have to be supported:

Manually specified information is not intrusive, but rarely up-to-date. Including such information is required if other techniques are not applicable.
Passive scanning has no direct impact on networks, but collected information is limited and access to all network segments is required.
Active scanning creates load in a network and on components, but provides more detailed information about entities and services in the network.
Network management protocols need to be available on investigated nodes, but reduce system's load and information requests are standardized.
Direct access to components, e.g. with *SSH*, delivers rich information, but requires appropriate access, to invoke applications, and interpret the output.
Agent-based information collection collects information just-in-time, but agents need to be deployed and run on the components.

(c) Deployment Process: The deployment process has to be *incremental*, so that the delta between the current and the target state is computed during setup and only required configuration steps are executed. Additionally, the deployment process has to be modular and extensible in order to cope with use case specific requirements. Therefore, the setup process has to be divided into small, self-contained steps. To be able to use the deployment process on different testbed architectures, the process itself needs to be independent from the underlying architecture as much as possible.

3 Related Work

To the best of our knowledge, no application to reproduce network environments exists. Therefore, we examine the two main components of INSALATA, namely information acquisition and testbed setup for deployment, separately. Next, we investigate network description languages as we need a suitable network model for INSALATA. Finally, we discuss network management protocols and frameworks to investigate appropriate implementation mechanisms to deploy the description of the network environment on the testbed.

Data Acquisition Applications are needed to obtain information about the network environment. Here, continuous monitoring is required and information from different sources needs to be fused. Additionally, tracking changes is a requirement. *IO-Framework* [6,19] does not support continuous monitoring and only supports intrusive collection methods. The *common Network Information Service (cNIS)* [1] utilizes static information and higher level services (SSH or SNMP) but does not include less invasive information collection techniques. *MonALISA* [7,11] and *PerfSONAR* [29] are not capable of continuously monitoring the network and detect changes. *OpenVAS* [23] is used to identify vulnerabilities within an infrastructure but has limited scanning capabilities. Single purpose tools like *Nmap* [20], *Traceroute* [2], or *xprobe2* [35] can be used to collect single aspects of the network environment but do not provide a holistic view. Dedicated network management protocols, like *SNMP* [8,21] or *Netconf* [27] can only be used to retrieve dedicated information from single network components, but do also not provide a complete view on the network.

Testbed Management Frameworks are used to orchestrate and control testbeds. All presented frameworks do not provide incremental setups but rebuild the designated network from scratch leading to higher effort within the setup process and manually configured changes get lost. Additionally, testbed orchestration and experiment execution are often tightly coupled. *vBET* [18] and *Laas-NetExp* [24] are both closed source, preventing to extend those frameworks. *VNEXT* [25] and *NEPTUNE* [5] do not provide the automated setup of basic network services, like DNS or DHCP. *Emulab* [34] or *DETER* [4] tightly couple the infrastructure setup and the experiment execution. This requires to rebuild the network after each experiment.

Network Description Languages and Ontologies can be used as information model. The related work within this field lacks in providing the information elements needed for a proper mirroring of network environments, especially in terms of reflecting the connectivity due to routing and the usage of firewalls. *IF-MAP* [3,30,31] is not capable of reflecting interfaces or routing information. The *target-centric ontology for intrusion detection* [32] does not provide a sufficient addressing scheme for elements nor reflect routing or firewalls. The *Network Markup Language (NML)* [14,15] provides a schema for exchanging network descriptions on a generic level, but does not provide concepts like network routing. The *Infrastructure and Network Description Language (INDL)* [13,17,28] extends NML, but is not capable of reflecting routes or firewall rules. Tcl-based format in *Emulab* and *ns-2* [34] is not capable of modeling networks explicitly resulting in verbose definitions for large networks.

Network Management Protocols and Frameworks can be used to setup and configure the descriptions in testbeds. To do so, the virtualized testbed has to be setup, e.g. router and hosts as virtual machines, and those components need to be configured appropriately afterwards, e.g. using adequate routing tables. Dedicated network management protocols, like *SNMP* [8,21] or *Netconf* [27]

can be used to configure components and request dedicated information in a standardized way. Both protocols do not provide built-in mechanisms to manage larger infrastructures as a whole. *Ansible* [9] is a push-based framework to configure larger infrastructures using so-called *playbooks*. Those playbooks need to be written or adapted for each configuration. Ansible can not be used directly for our approach, but is suitable as an important building block. *Puppet* [26] is a pull-based framework for infrastructure configurations. As the testbed is reconfigured in irregular intervals, a pull-based mechanism is not suitable.

4 Approach and System Design

INSALATA consists of the Collector and the Deployment Component as its two main components:

The **Collector Component** is responsible for collecting and fusing information of the network environment and the current state of the testbed into a descriptions (see Sect. 6).
The **Deployment Component** manages configuration changes and the automated setup process on the testbed (see Sect. 7).

Both components utilize the same *information model* to structure the information about the network environment (see Sect. 5) and are managed and orchestrated by a central controller, the *Management Unit*. The system architecture of INSALATA showing these basic components and their interaction is depicted in Fig. 1.

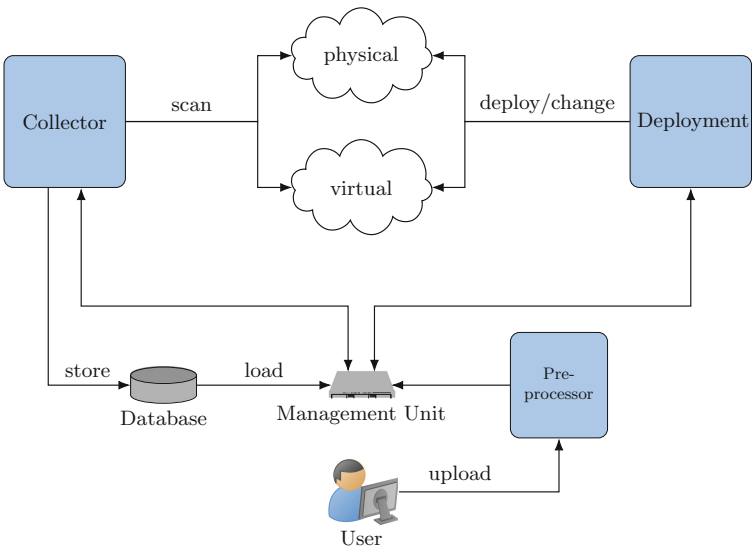

Fig. 1. System overview of INSALATA showing basic components

The Collector acquires information about the network environment and is used to generate a description for the testbed. It maintains the current state of the monitored network. Here, the collection process can be done continuously whereas configuration changes can be tracked and stored with a timestamp in a database. This approach allows to rebuild a network at each point in time.

The Deployment Component utilizes a description that is either obtained by the Collector or provided by the user. Based on this, the Deployment Component configures the testbed to reproduce a network environment. To ease writing descriptions, a *Preprocessor* is utilized, replacing missing but calculable values in the description and checking the it for its validity.

5 Description of the Information Model

To be able to reflect a network environment in a testbed, a formal description of this network is required. This description needs to contain *information elements* discussed in Sect. 2. Each information element needs to be leviable from the network environment in an automated manner and has a unique *identifier*. The proposed information model is shown in Fig. 2. An information element is represented as box, the identifier of each element is underlined and additional attributes describing the information element are listed. For FirewalRules and Routes a combination of attributes is used as identifier. Relations between information elements are denoted as arrows in between and additionally denote their cardinality.

The main information element is the *Network Component* representing a node in the network environment, e.g. hosts, routers or switches, and are further specified by the *Template* attribute. A Network Component is equipped with certain *Disks* and *Interfaces*. Interfaces are needed to interconnect Network Components in different kinds of networks, e.g. *Layer 2 Networks* or *Layer 3 Networks*.

Depending on its functionality, a Network Component can maintain *Routes* or *Firewall Rules*. Those express the connectivity between Network Components. The latter can be represented as raw dumps (*Firewall Raw*) or in a simplified format (*Firewall Rule*) to ease transformation between different firewall applications as proposed in [10]. As a simplification is not free of information loss, the raw information is stored additionally.

To support large-scale test environments consisting of multiple servers, a Network Element is associated with a *Location* specifying the testbed server the Network Component is emulated on. In case of a description reflecting the network environment, the Location is set to *physical*.

Another important concept of the information model is the *Service* element. This information element is used to reflect basic network services, like *DNS* or *DHCP*. A Service can be further specified by adding a *Product* and a *Version* to allow a high accuracy. Additionally needed services can be added to the information model using inheritance, allowing use case specific applications.

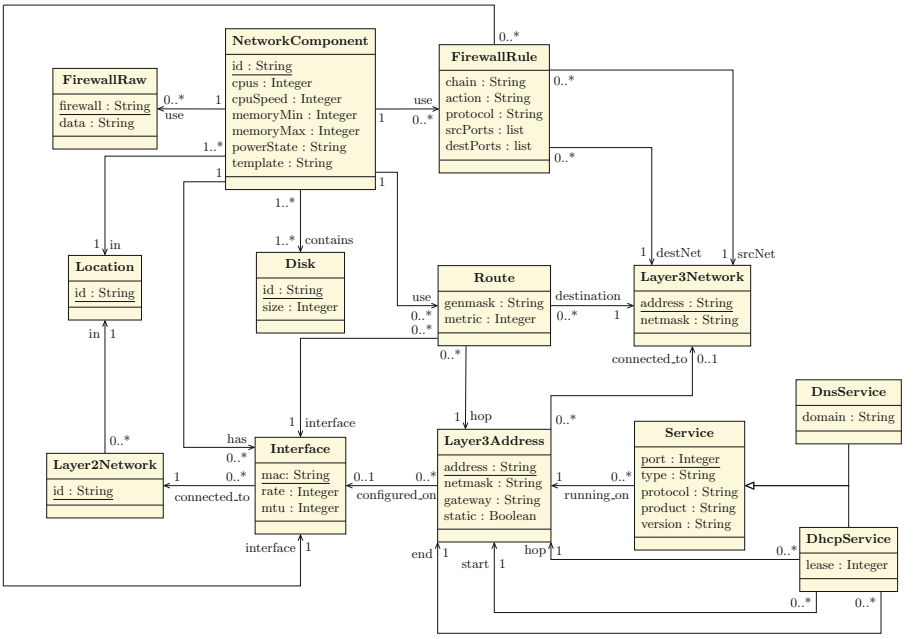

Fig. 2. Information model of INSALATA

6 Information Collector Component

The Collector Component is capable of managing multiple *Environments* describing multiple networks to be monitored. The overview of INSALATA's information *Collector Component* is depicted in Fig. 3.

For each Environment, multiple *Collector Modules*, employing a particular information collection technique as described in Sect. 2, can be configured to collect the required information. A Collector Module obtains information about at least one information element and possible relations between information elements. Therefore, modules have to obtain the unique identifier of an information element. The collected information elements are handed over to the Environment fusing all obtained information into a comprehensive graph describing the network. Here, we assume that a module delivers no false, but potentially incomplete information. This modular approach has the advantage that different, specialized information collection techniques can be combined resulting in a more detailed and more precise view on the network environment.

To fuse information, the Collector utilizes the identifier of each object and the type of the information element. Objects with the same identifier and of the same type are treated as the same object. Each Collector Module passes discovered objects to the Collector. Attributes and relations are fused together in case multiple modules report the same objects.

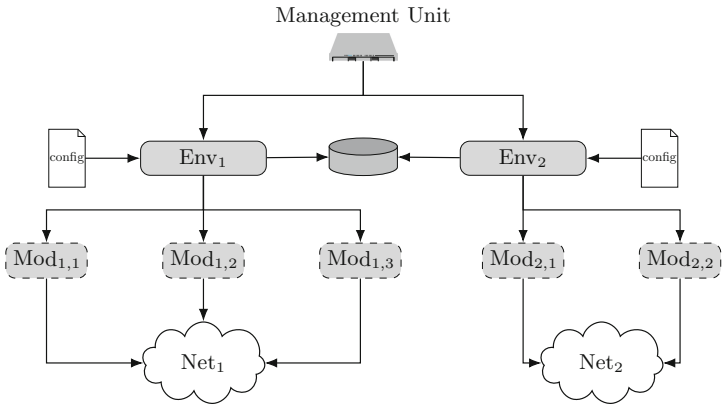

Fig. 3. Information Collector Component of INSALATA

Another challenge is to manage existing information elements in the model becoming obsolete. Therefore, a deletion scheme needs to be implemented within an Environment. Each information element in an Environment is equipped with timers for each Collector module. Each time, an information element is delivered by a module, the module specific timer is updated. If all timers in the list expire, the information element and its relations are deleted from the Environment. In addition, each module can actively set its own timer to zero, if it is capable to determine the non-existence of an element.

To be able to recreate an environment at any (observed) point in time, we track the network environment's state over time. Whenever an information element or relation is modified, i.e., is added, deleted, or updated, we save this delta as an event to the database. Such events contain the type of change, the information element and its properties. Within an Environment, only the current state of the network description is maintained.

7 Infrastructure Deployment Component

The Deployment Component executes the following steps: (a) determine required configuration steps using the current testbed state and the given description, (b) determine a correct *execution plan* how to achieve the given description, and (c) deploy the changes on the testbed following the computed execution plan. The overview of the execution flow of INSALATA's Deployment Component is depicted in Fig. 4.

First, the Deployment Component has to determine *what* needs to be changed. Therefore, we need the current state of the testbed in the form a description as *Description* D_2 that can be determined using the Collector Component. Additionally, we need the target state in the form a description as *Description* D_1 provided from the Collector or the user. The Deployment

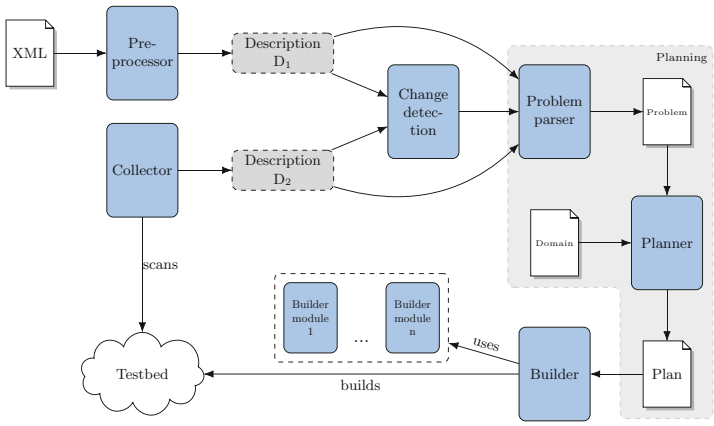

Fig. 4. Execution flow of INSALATA's Deployment Component

Component uses a *Change Detection Module* to detect added, removed, and updated information elements in the delta between these states.

To determine *how* to change the testbed, we use *automated planning and scheduling* from the domain of artificial intelligence [16]. A planning problem is described using a dedicated planning language such as the *Planning Domain Definition Language (PDDL)*. PDDL separates the planning problem into a *domain* description, describing the problem domain, and a *problem* description, describing an instance of the problem [12]. A PDDL domain description describes the *objects*' *types*, *predicates* (i.e., properties), and *actions*. Each action has a definition of objects it is applicable to, *preconditions* that have to be fulfilled, and an *effect* altering the predicates of objects. With INSALATA, we provide a domain description for our information model and steps as PDDL actions necessary to setup a testbed. A detailed overview on the steps we defined and their interdependencies can be found in the domain description provided with the implementation. The PDDL problem description describes all objects, their type and their *initial state*. The problem specifies the *goal state* of all objects by giving their desired predicates [22] inside a goal section. While the domain file is static, the problem file depends on the current and target state and is computed each time a description is deployed on the testbed.

The computed changes, the current testbed state, and the target state are given to the *Problem Parser Module* computing a PDDL problem description. This description is given to the *Planner Module*, an automated planning and scheduling solver computing an execution plan containing the correct order of actions that will bring the testbed from the current into the target state. The execution plan is passed to the *Builder* to execute the steps on the designated testbed. Implementations of these steps are provided by architecture-specific *Builder Modules* since the implementation of such steps is different for different testbed architectures and objects. The Builder uses meta-data associated with the Builder Modules and the objects to identify the correct implementation.

This allows us to dynamically add new implementations, e.g. for new operating systems or services, without changing the Deployment Component.

8 Implementation

INSALATA is written in Python 3 and is available in INSALATA's GitHub repository[1]. A detailed code documentation is available on[2].

The *Management Unit* can be controlled using a XML-RPC client communicating with INSALATA. The presented information model is reflected using object-oriented Python classes.

Besides the *Collector Component* itself, we provide the following *Collector Modules*: (1) a *XEN module* using to collect information from environments using XEN virtualization, (2) an *XML module* for manually provided information, (3) a *Tcpdump module* for passive network scanning using `Tcpdump`, (4) an *Nmap module* for active network scanning using `Nmap`, (5) an *SNMP module* to retrieve information from nodes using `SNMP`, (6) an *SSH module* to retrieve information from nodes via `SSH`, and (7) a *Zabbix module* for agent-based information collection with the Zabbix network monitoring system. Some modules have limited functionality, meaning that not all information possible to collect is implemented, e.g. the SSH module does not collect firewall rules. New collector modules can be added to INSALATA to cover the desired scanning environment.

We integrate *fast-forward*[3] as a planner into INSALATA's *Deployment Component*. The domain file describing the setup process of a testbed we provide, is given in `PDDL`. The required problem files are generated for each setup individually in an automated manner. We provide a framework allowing to add new *Builder Modules* in an easy way. Here, we utilize Python annotations to determine the most suitable Builder Module for a given step within the determined plan and the configured object. Besides provided Builder Modules to setup the basic topology on XEN (hosts, routers, layer 2 networks), we utilize Ansible for additional configurations (routing, firewalling, DNS, and DHCP).

9 Case Study: Chair's Teaching Infrastructure – iLab

To show INSALATA's applicability in practical scenarios, we assesed INSALATA in a case study. For this case study, we use a setup adapted from the lab course *iLab*[4] offered by the TUM's Chair of Network Architectures and Services. The iLab is a course to teach student's practical skills in administering network setups and configurations for different scenarios using real hardware. A typical setup students have to work with during an iLab course is shown in Fig. 5. This setup consists of two Cisco and a Linux router, and four host residing in different private networks. In our case study, our goal is to reproduce this network environment in a network testbed using XEN virtualization.

[1] https://github.com/tumi8/INSALATA.
[2] https://insalata.readthedocs.io/en/latest/index.html.
[3] http://www.fast-downward.org/ObtainingAndRunningFastDownward.
[4] https://ilab.net.in.tum.de.

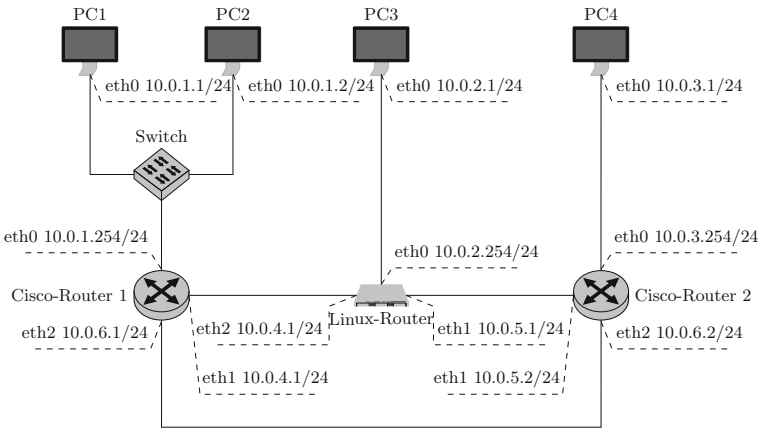

Fig. 5. Typical infrastructure setup within the iLab course

We provide the IP addresses of all routers to the INSALATA system as a starting point using manual input in form of XML files. All routers and hosts are configured to allow INSALATA to access the systems using SSH and SNMP.

In the first phase, we obtain the required description of the network environment using our Collector Component. To expand our infrastructure information using our passive *Tcpdump* Collector Module, we generate traffic on the involved hosts. Using the *SNMP* and *SSH* Collector Modules, missing interfaces, MAC addresses, and routing information is obtained from routers and hosts. Using theses Modules, we are able to reflect the network environment shown in Fig. 5 as description.

In the second phase, we use our Deployment Component to reproduce the obtained description into our virtualized testbed using XEN with the *xapi* toolstack. INSALATA computes an execution plan to setup the testbed from scratch in 0.252 s. The resulting execution plan consists of 92 steps, including setting up virtual machines and networks, configuring interfaces and deploying routes. Each step is executed in sequence and no parallelization is done. The total time required to setup the testbed and configure it is 42 min 32 s. Figure 6 visualizes the setup and configuration process in regard to its execution time.

The most time-consuming tasks during the setup process are the creation of new virtual machines, which happens at the beginning. The reason for this is that here new virtual machines have to be cloned from the respective template, including copying hard disk images and required reboot operations. Configuration steps like creation of virtual networks and interfaces are done nearly instantaneously. After the setup of our description, we validated the correctness of our setup using manual inspection, *Ping* and *Traceroute*.

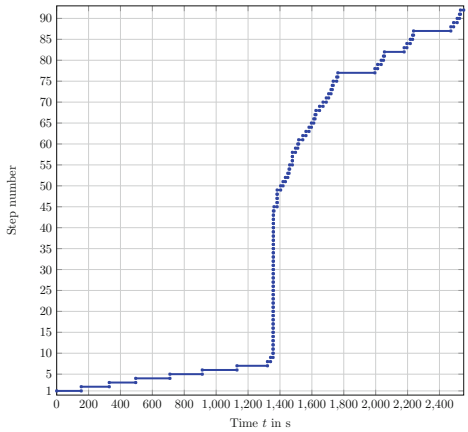

Fig. 6. Time distribution of setup steps in the iLab case study

10 Conclusions and Future Work

In this work we present INSALATA, a system capable of reproducing network environments in network testbeds. INSALATA enables network operators and researchers to test and analyze new security features and general configuration changes in a separated test environment before deployment in operational networks. To be able to formalize network environments, INSALATA utilizes an information model particularly crafted for representing network topologies, entities, and services in *descriptions*. To obtain the required information from network environments, we support a modular *Collector Component* automatically assessing networks and fusing information from different *Collector Modules*. The *Deployment Component* provides automated planning and scheduling to instantiate descriptions onto a physical or virtualized testbeds. Within our case study, we show the applicability of our approach reproducing a real world network environment with several routers and hosts onto a virtualized testbed.

To further improve INSALATA, we will continue our work in this field and on INSALATA and highly appreciate feedback, improvements, and extensions from the community. To extended INSALATA's capabilities, additional Collector and *Builder Modules* can help to obtain additional properties from the network environment, such as user information or generic service configurations from hosts. Existing Collector Modules can be extended to obtain more information using existing assessment techniques, such as firewall information using SSH. To reproduce network environments more realistically, Builder Modules to support Microsoft Windows and additional network services, like mail or ftp are beneficial. One of our main goals is to make the deployment process more efficient by parallelizing the execution of the setup plan. Since INSALATA only provides mechanisms to setup and orchestrate a testbed, we aim to integrate our experiment execution framework GPLMT [33] into the INSALATA system.

Acknowledgments. This work has been supported by the German Federal Ministry of Education and Research (BMBF) under support code 16KIS0538 (DecADe).

References

1. GEANT2 common Network Information Service (cNIS) Schema Specification. http://www.geant2.net
2. traceroute(8) - Linux man page. linux.die.net/man/8/traceroute
3. Ahlers, V., Heine, F., Hellmann, B., Kleiner, C., Renners, L., Rossow, T., Steuerwald, R.: Integrated visualization of network security metadata from heterogeneous data sources. In: Mauw, S., Kordy, B., Jajodia, S. (eds.) GraMSec 2015. LNCS, vol. 9390, pp. 18–34. Springer, Cham (2016). doi:10.1007/978-3-319-29968-6_2
4. Benzel, T.: The science of cyber security experimentation: the DETER project. In: Proceedings of 27th Annual Computer Security Applications Conference, pp. 137–148 (2011)
5. Bifulco, R., Stasi, G.D., Canonico, R.: NEPTUNE for fast and easy deployment of OMF virtual network testbeds [Poster Abstract] (2010)
6. Birkholz, H., Sieverdingbeck, I., Sohr, K., Bormann, C.: IO: an interconnected asset ontology in support of risk management processes. In: 7th International Conference on Availability, Reliability and Security (ARES), pp. 534–541 (2012)
7. Carpen-Amarie, A., Cai, J., Costan, A., Antoniu, G., Boug, L.: Bringing introspection into the BlobSeer data-management system using the MonALISA distributed monitoring framework. In: International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), pp. 508–513 (2010)
8. Case, J.D., Fedor, M., Schoffstall, M.L., Davin, J.R.: Simple network management protocol (SNMP). RFC 1157, IETF, May 1990
9. DeHaan, M.: Ansible (2012–2016). https://www.ansible.com/
10. Diekmann, C., Hupel, L., Carle, G.: Semantics-preserving simplification of real-world firewall rule sets. In: 20th International Symposium on Formal Methods, pp. 195–212 (2015)
11. Dobre, C., Voicu, R., Legrand, I.: Monitoring large scale network topologies. In: IEEE 6th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), vol. 1, pp. 218–222 (2011)
12. Fox, M., Long, D.: PDDL2.1: an extension to PDDL for expressing temporal planning domains. J. Artif. Int. Res. **20**(1), 61–124 (2003)
13. Ghijsen, M., van der Ham, J., Grosso, P., Dumitru, C., Zhu, H., Zhao, Z., de Laat, C.: A semantic-web approach for modeling computing infrastructures. Comput. Electr. Eng. **39**(8), 2553–2565 (2013)
14. van der Ham, J., Dijkstra, F., Apacz, R., Zurawski, J.: Network Markup Language Base Schema version 1 (2013)
15. van der Ham, J., Dijkstra, F., Lapacz, R., Brown, A.: The Network Markup Language (NML): A Standardized Network Topology Abstraction for Inter-domain and Cross-layer Network Applications (2013)
16. Hoffmann, J.: Everything you always wanted to know about *Planning*. In: Bach, J., Edelkamp, S. (eds.) KI 2011. LNCS, vol. 7006, pp. 1–13. Springer, Heidelberg (2011). doi:10.1007/978-3-642-24455-1_1
17. van der Ham, J.J.: A semantic model for complex computer networks: the network description language. Ph.D. thesis, University of Amsterdam (2010)

18. Jiang, X., Xu, D.: vBET: A VM-based emulation testbed. In: Proceedings of ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research, pp. 95–104 (2003)
19. Lorenzin, L., Cam-Winget, N.: Security automation and continuous monitoring (SACM) requirements. Internet-Draft draft-ietf-sacm-requirements-15, Internet Engineering Task Force (2016)
20. Lyon, G.: nmap(1) - Linux man page (2015)
21. McCloghrie, K., Rose, M.: Management information base for network management of TCP/IP-based internets. RFC 1156, IETF, May 1990
22. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL - the planning domain definition language (1998)
23. OpenVAS: Project Homepage. http://www.openvas.org/
24. Owezarski, P., Berthou, P., Labit, Y., Gauchard, D.: LaasNetExp: a generic polymorphic platform for network emulation and experiments. In: Proceedings of 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities, no. 24, pp. 1–9 (2008)
25. Pisa, P.S., Couto, R.S., Carvalho, H.E.T., Neto, D.J.S., Fernandes, N.C., Campista, M.E.M., Costa, L., Duarte, O., Pujolle, G.: VNEXT: Virtual network management for Xen-based testbeds. In: International Conference on the Network of the Future (NOF 2011), pp. 41–45 (2011)
26. Puppet Labs: Puppet (2005–2016). https://puppet.com/
27. Enns, R., Bjorklund, M., Schoenwaelder, J., Bierman, A.: Network configuration protocol (NETCONF). RFC 6241, IETF, June 2011
28. Taketa, T., Hiranaka, Y.: Network design assistant system based on network description language. In: 15th International Conference on Advanced Communication Technology (ICACT), pp. 515–518 (2013)
29. Tierney, B., Metzger, J., Boote, J., Boyd, E., Brown, A., Carlson, R., Zekauskas, M., Zurawski, J., Swany, M., Grigoriev, M.: perfSONAR: instantiating a global network measurement framework. In: SOSP Workshop on Real Overlays and Distributed Systems (ROADS 2009). ACM (2009)
30. Trusted Network Connect Work Group: TNC IF-MAP Bindings for SOAP, Version 2.2, Revision 10 (2014)
31. Trusted Network Connect Work Group: TNC MAP Content Authorization, Version 1.0, Revision 36 (2014)
32. Undercoffer, J., Pinkston, J., Joshi, A., Finin, T.: A target-centric ontology for intrusion detection. In: Proceeding of 9th Workshop on Ontologies and Distributed Systems, pp. 47–58 (2004)
33. Wachs, M., Herold, N., Posselt, S.-A., Dold, F., Carle, G.: GPLMT: a lightweight experimentation and testbed management framework. In: Karagiannis, T., Dimitropoulos, X. (eds.) PAM 2016. LNCS, vol. 9631, pp. 165–176. Springer, Cham (2016). doi:10.1007/978-3-319-30505-9_13
34. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. pp. 255–270 (2002)
35. Yarochkin, F., Arkin, O., Kydyraliev, M.: xprobe2(1) - Linux man page