2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)

The Need for unified Testbed Management across multiple Teams and Stakeholders in a large scale Telecom Integration Context

Nikolay Tcholtchev[1], Steven Ulrich[1], Faruk Catal[1], Wojciech Konitzer[1], Ramon Barakat[1], Andreas Hoffmann[1] and Ina Schieferdecker[1]
[1]Fraunhofer Institute for Open Communication Systems (FOKUS)
Kaiserin-Augusta Alee 31, 10598, Berlin, Germany
{firstname, lastname}@fokus.fraunhofer.de

Abstract — **The quality assurance of large scale integrative systems often requires complex testbed environments and simulations that allow to test the overall functionality and enables various experiments towards systematically verifying the realization of the identified user and system requirements. Thereby, an integration setup and resulting activities lead to another level of quality assurance, whereby the integrator deals with the quality examination of the single components and their integrative interplay according to a set of overall system and user requirements. In such context, it is often the case that the testing activity is conducted by various partners (e.g. single companies and legal entities) with complementing know how required for specific sub-tasks - e.g. PKI, chip cards, special network protocols, firewall, security architectures, and penetration testing. This leads to the emergence of a large number of proprietary testbeds focusing on specific aspects resulting in the lack of a unified testbed configuration, versioning and technological foundations (e.g. operating system, network stack implementations, hypervisor technology …). In this paper, we present our experiences drafted from a large scale industrial project with 600-700 requirements relating to a critical eHealth infrastructure within a telecom provider context. Thereby, various sub-contractors had to be unified in their approach to testbed management in order to achieve reproducible and traceable (with respect to system requirements) test results based on a test architecture accommodating various quality assurance activities (unit testing, development tests, component testing, integration testing, security testing …). We gradually analyze the project situation with respect to testbed management and argue on the need for unified testbed management across multiple teams and stakeholders in a large scale telecom integration setup. Subsequently, we propose possible solutions and conduct a series of experiments highlighting the advantages of the proposed approach and belonging solution.**

Keywords — docker, container, virtualization, eHealth, telecom, networking, component testing, black box testing, system integration

I. INTRODUCTION

Traditionally, testing is either deeply embedded in the development processes (e.g. unit testing, module testing) or takes place on a higher level of component matureness (e.g. integration testing, certification tests, interoperability testing …). This is especially valid for ICT products which are developed in the scope of a single company or legal entity. However, telecom operators normally do not develop a single product from scratch. Mostly, they have to compile a complex distributed environment based on products and components they acquire from multiple suppliers. Typical examples are given by large scale telecom and Internet networks, where various suppliers provide routers, switches, multi-media gateways, mobile core network components (e.g. GGSN, MSC, HLR …), CDN (Content Distribution Network) servers/brokers etc.

In general, the level of matureness is quite high for products of telecom suppliers. Hence, the testing which normally takes place on network operators' site, is mostly related to the interoperability of the selected components - e.g. OSPF/RIP/ISIS interoperability between routers from different vendors (e.g. between Cisco and Huawei), and to special features/extensions of the products, which are especially developed for the intended integrated solution the network provider works on.

Nevertheless, it might happen that the telecom integrator decides to develop a component on its own, as happened to be the case in a large scale project for an advanced eHealth infrastructure. Given the fact that telecoms are normally integrators, they rarely possess large and highly experienced development and quality assurance teams. Hence, relevant know-how has to be acquired on the free market, i.e. sub-contractors need to be appointed. This naturally leads to a significant number of players and stakeholders, with complicated legal situations, in the scope of a highly complicated technological landscape – the eHealth related product in question encompassed around 600-700 requirements of various complexity spanning over the different layers of the TCP/IP (and ISO/OSI) stack and strongly depending on aspects such as security, privacy, confidentiality, PKI, secure NTP and DNS (i.e. DNSSEC), VPN-Tunnels (i.e. IPsec) and various application layer proxies (e.g. HTTPS, OCSP …).

Due to the above descriptions and considerations, it easily comes to a situation where multiple test teams are engaged across different stages of the testing process. The stages are given by:

- Development tests – i.e. unit and module testing

978-1-7281-3925-8/19/$31.00 ©2019 IEEE
DOI 10.1109/QRS-C.2019.00084

- Model-in-the-loop and hardware-in-the-loop testing by switching between simulations - on the interfaces of the SUT (System Under Test) - and real hardware (on the interfaces of the SUT) thereby detecting implementation errors on early stages
- Product test - quality assurance/acceptance testing on top of the development test
- Security testing – i.e. penetration and certification testing for national agencies
- Component test of the integrator involving multiple teams with various capabilities
- Integration and interoperability testing across the overall infrastructure

Thereby, the teams are embedded in the above mentioned highly complex legal and technological circumstances – i.e. technical information flows slowly and has to pass multiple management levels whilst the technological challenges require the instant and efficient know-how and technical exchange across the teams. In particular, one serious issue emerged during the multi-stake holder test approach: **the synchronization of the test environment (i.e. testbed artefacts/images) across multiple stake holders**, which is also the scope of the current discussion.

The test environment for the eHealth infrastructure in question consisted of several virtual machines which were initially setup and distributed across the test teams. Thereby, each test work place was equipped with the full set of virtual machines and each tester was utilizing the full set of VMs within each team. In addition, the developers were using the virtual machines, in order to advance the firmware development. Hence, immediately after the initial VM design phase, a large number of testbeds was distributed without any means for proper synchronization. The testbed VMs were based on different traditional hypervisors, such as Virtual Box, KVM or VMware. Hence, changes which were made to the images needed to be clearly communicated across the testing teams, i.e. a **change log** had to be taken care of and communicated among the involved test stages and teams. However, given the complex legal setup of the project, the communication turned difficult and error prone. Hence, a way **to efficiently manage, synchronize and distribute the testbed images across teams and work stations was required**.

The above described situation has led to the evaluation of different possibilities for enabling testbed management, such as SVN, GitLab, Docker, DockerCompose, Vagrant, Ansible, as well as traditional bash scripting and ssh/sshpass based solutions. The current paper describes the experiences that were gained in such a multi-stakeholder testing setup with respect to the **Unified Testbed Management across Multiple Teams and Stakeholders in a large scale Telecom Integration Setup**. In addition, some very positive side effects are in the scope, such as testbed stability and improved root cause analysis for failed test cases, accelerated test execution, efficient exchange of test environments and improved tickets/issue analysis and failed-test-case resolution.

The rest of this paper is organized as follows: Section 2 presents a number of related technologies and approaches. Section 3 describes the project context relating to a large scale eHealth infrastructure developed in an integrative manner by a large telecom service provider. Section 4 provides a detailed analysis of the initial problems regarding testbed management, which is the base for the following section 5 where a concrete solution approach is proposed and analyzed in terms of how it addresses the identified initial problems. Section 6 presents a number of numerical results showing how our solution improved the testbed stability and the test execution process as a whole. Finally, a summary is presented and a set of conclusions are drawn.

II. Related Work

The topic handled in this paper is fairly unusual given that integrators normally do not develop components but setup the distributed solutions thereby strongly relying on the quality assurance of the suppliers/vendors. Thereby, the integrator normally focusses on end-2-end integration tests based on the real hardware and does not really need simulation based testbeds and playgrounds. For managing the end-2-end user acceptance tests various types of software is on the market, including HP-Quality Center [2], Jira [4] and Testrail [3]. Thereby, special testing and demonstration labs are established showing the overall capability of distributed system, with the various 5G test fields across the world being typical examples [5][6][7]. Furthermore, the living labs established in European research projects can be seen of typical examples for integrator testbeds for end-2-end user scenarios.

With respect to testbeds, standardization and certification bodies tend to provide unified testbeds which are made accessible to vendors in order to test their products without allowing the configuration chaos experienced in the project in question. Typical such efforts are given by the IPv6 Ready Logo program and its belonging testbeds [8][9][10] as well ETSI [12], and gematik [13] with their belonging unified test suites and test environments. Thereby, the testbeds are either managed in the form virtual machines based on belonging supervisors, e.g. VirtualBox [14][15], Qemu [16], KVM [17], VMware [18][19] or Xen [20]. Furthermore, the Linux container technology is also of paramount importance for this paper - with docker [21][22][23][24] and LXC [23] as most prominent representatives of the container type of virtualization. Moreover, testbed management is often conducted by utilizing special middleware for steering the testbed components such as OpenShift [25], Kubernetes [23], OpenStack [26] and OpenNebula [27]. Thereby, concepts from the area of network management and Software Defined Networking (SDN) [28] (e.g. OpenFlow [29]) can be very helpful to efficiently manage the virtual machines and testbed components.

To give an example for playgrounds and development environments, which are available as testing services: Fraunhofer FOKUS provides different environments for end-2-end integration and interoperability testing such as the Interoperability Lab [31], IPv6 Testing and Network

Simulation Lab [30] as well as diverse 5G [5] and Machine-2-Machine playgrounds and testing environments.

With regards to test strategy, different guidelines are available, for instance in the form of IETF RFCs [33] or as test concepts and processes such as IEEE 829 [33]. Furthermore, [32] provides some key design patterns for test approaches and test automation, which were already applied in the current context [1]. Furthermore, the topics [35][36][37] of DevOps, Continuous Delivery and Continuous Integration constitute a new modern wave of integrated testing approaches and belonging team culture, where the overall software system is continuously being integrated and tested with the goal of quick efficient and qualitative releases and software delivery. Thereby, virtualization and container technology play a key role. Furthermore, the methodology presented in this paper has some common points with the above approaches and can be easily integrated in a DevOps context spanning the overall telecom integration setup.

III. Project Context

The project context of our considerations is given by an eHealth service to be implemented on a large scale by one of the main network and service providers in Germany. Thereby, components originating from various vendors need to be integrated and later on to operate seamlessly within hospitals, doctors' premises and emergency situations. The components include front end applications (web, mobile and desktop), access routers, firewalls, unified thread management solutions, various types of VPN boxes (mainly IPSec layer 3 boxes), a large variety of trusted networking services such as DNSSEC, NTP, QoS (mainly DiffServ), HTTP-Proxies, configuration repositories as well as security related services for distributing cryptographic material and validating certificates (e.g. OCSP). Furthermore, different chip cards and belonging readers were also tested and integrated, or required to be simulated, in order to evaluate neighboring components and complex integrative end-2-end scenarios.

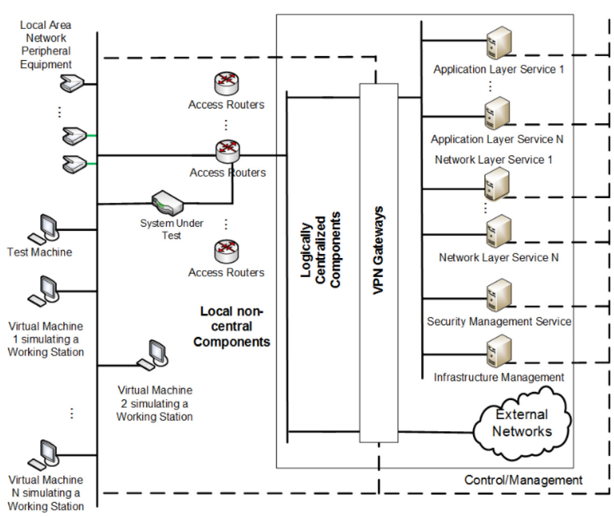

FIGURE 1: SKETCH OF THE TESTBED SPECIFICATION AS DESCRIBED IN [1]

Figure 1 illustrates a sketch of the testbed that was required in the course of the project execution. One can clearly see the local area network segment embedding aspects such as frontend user interfaces to be operated within hospitals and doctors' premises. Furthermore, the access network component denoted as an SUT stands out as integrative module between the Internet/telecom network and the backend data center architecture providing various of the above mentioned services and functionalities as well as gateways and secure communication interface to external networks.

In [1], we have already analyzed some of the drawbacks of the initially undertaken testing approach within this project. The identified issues violated some basic test automation patterns [32] such as COMPLEX ENVIRONMENT, INEFFICIENT FAILURE ANALYSIS, HIGH ROI EXPECTATIONS and NO INFO ON CHANGES. These issues were gradually fixed in the course of the project duration, whereby the current paper describes important aspects relating to the issues of COMPLEX ENVIRONMENT and INEFFICIENT FAILURE ANALYSIS, provided that the complexity and diversity of the testbed implementation has led to unmanageable situation hindering the failure analysis process and turning into a serious obstacle for the overall system certification.

It is important to emphasize that for a long time the abstract testbed from Figure 1 with its belonging implementations was used for various phases of the testing process – e.g. unit testing, module and components testing, security/penetration testing, load- and performance testing, as well as for the testing of different components - thereby integrating real components with simulations – without having any testbed management approach across multiple involved teams, partners and stakeholders. This has led to a fragmenting of the testbed versions whereby even within one team, a number of different testbed configurations were circulating and were utilized in the course of test execution. The resulting problems are systematically analyzed in the coming section.

IV. Problem Analysis

Table 1 sums up all the key risks and issues that were observed without following the path of a unified testbed management. Instead of a unified testbed management, we based our activities on a set of hypervisor based virtual machines that were distributed across the test and development teams and updated occasionally in case any communication has taken place. Indeed a number of serious issues were encountered that range from non-comparable test results, false positives and lead to an extremely inefficient handling and correction of defects and problems in the belonging SUTs. More details as well as the observed frequency of the issues are given in Table 1.

Table 1: Risks, Analysis and belonging Frequency

Issue/Risk	Analysis/Description	Frequency (rare/often/ very often)
	We very often encountered	very often

No_info_on_changes	situations were changes were conducted on the tesbed virtual machines from within one of the involved test and development teams. Correspondingly these changes were not communicated to the other teams which led to a divergence of the used testbed versions across the various teams and stakeholders.	
Proprietary_testbed_configurations	The missing communication between the (often) competing teams has implied a large number of proprietary configurations which were extremely difficult to synchronize across the difficult teams and have led to a chaotic situation w.r.t. aspects such as test result reproducibility etc.	very often
Inefficient_synchronization_on_specification_updates	The overall system specification has been occasionally changing (around two times per year) which required adaptations in the testbed environment. Provided the separation of the teams and the lack of a unified process, the adaptations were conducted in different ways, which has finally led to large proprietary deviations that could be traced back to conflicting interpretation of the specification changes.	rare
Non_comparable_test_results	The proprietary and deviating testbed variations have led to test results which were not comparable across the various teams and stakeholders. This resulted in costly and time consuming discussions paired with corresponding debugging sessions.	often
Inefficient_handling_of_defects	All issues/risks described hitherto have led to a highly inefficient handling of failures and belonging tickets/defects. The failed test results in one team were very often not reproducible within the environment of the other teams, leading to costly and time consuming discussions and controversies regarding the interpretation of the specifications and the test results.	very often
Instabilities_in_testbed_handling	The differences in the testbed versions - even within the same team[1] - have led to many instabilities and differences in the way the testbed components were handled within the test scripts. Some test scripts could only be executed on particular work stations and their results and execution flows were very different due to the testbed configuration chaos.	often
Incompatible_cryptographic_material	Another aspect of incompatibility having its origins within the testbed problems relates to the incompatible cryptographic material (certificates, Certificate Revocation Lists, DNSSEC keys …) across the different testbed versions. These cryptographic artefacts were diverging in various details such as the utilized cypher-suites, the certificate chains etc. In many cases this has led to incompatible diverging test results in different environments.	rare
False_positives	All the described differences have sometimes led to false positive test results in cases when a PASSED result got wrongly accepted in the overall discussion among the teams. In such situations, the responsible test team has wrongly configured its proprietary testbed based on a misunderstanding of the technology or the specification.	rare

V. Proposed Solution

The solution emerging from the above identified risks is based on the utilization of container technology instead of traditional hypervisor technology for the sake of testbed management. Thereby, the widely accepted docker container

[1] We even observed that different tester or test automation workstations within the same team were experiencing severe difference with regard to their testbed configurations.

solution was used, in order to setup an initial version of the required complex testbed environment.

Docker uses a so-called Linux base image that is established as the basic operating system configuration for the docker containers running on top. The specific configurations for each docker image (be it the NTP, DNS, VPN-gateway, OCSP responder …) are put in place in the form of a file system structure with belonging configuration files (e.g. */etc/ipsec.conf*) allowing to load each container with its own specifics without burdening the host with regard to managing a whole virtual node (for each of the testbed components) with all its overhead for restart and specific configurations. Furthermore, the overall set of docker-nodes was glued together into an integrated testbed by the means of a yaml-configuration file that allowed to describe the network interfaces (on link and network level, i.e. MAC and IP addresses) and to connect them correspondingly to an overall test environment for the various phases of testing as well as for the various components of the integrated eHealth solution as an SUT. This has brought a decisive advantage over the hypervisor approach exposing the testbed structure and the specific configurations in an easy to handle text form for all the involved testbed components. Hence, such text artefacts can be easily shared and managed over corresponding sharing and versioning infrastructure as the one described in the coming paragraph.

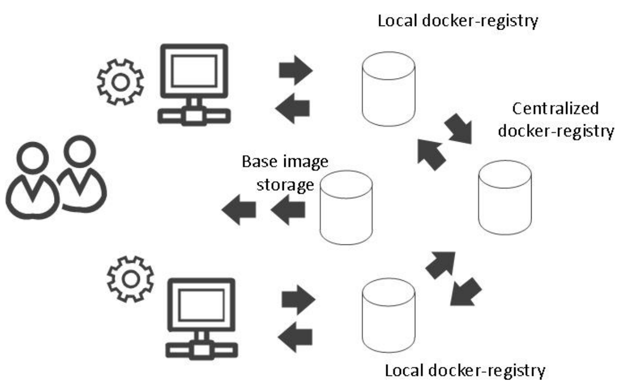

FIGURE 2: OVERALL VIEW OF THE PROPOSED AND IMPLEMENTED SOLUTION FOR UNIFIED TESTBED MANAGEMENT

The overall collaboration process relating to the unified testbed management across multiple teams is illustrated in Figure 2. Within this context, the Linux base image is stored in a centralized storage (e.g. FTP, WebDAV or NFS) and can be correspondingly adopted by all involved partners – an activity, which should not be considered very frequently, since the base image contains fundamental operating systems configurations meaning that most of the specific lightweight configurations are expected within the docker images. The docker images with their belonging text file configurations and file system structure are managed within an eco-system of local docker-registries and a centralized docker-registry on top, which enables the synchronization across multiple stakeholders and multiple teams. The docker-registries largely resemble the well-known gitlab structure and mechanisms, including familiar commands and processes such as *merge*, *push*, *pull* etc. Furthermore, the *docker-compose* tool is used to compile a local binary version of the overall set of containers, which can be efficiently executed on the local host where the execution for a predefined set of test cases takes place. Thereby, docker-compose can be seen as the compiler assembling the testbed and enabling the consequent test procedures.

The risks identified in the previous section with their belonging mitigation and observed results are depicted in Table 2 thereby rounding up the picture regarding the impact of our identified solution.

Table 2: Identified Risks, their Mitigation and belonging observed Results

Issue/Risk	Mitigation	Result
No_info_on_changes	Based on the docker images and the established exchange infrastructure (gitlab, docker-registries …), changes to the belonging network and configurational setup were easily communicated between the team members and stakeholders.	solved
Proprietary_testbed_configurations	The testbed configurations were continuously synchronized across the different teams based on the docker-files and the centralized repositories accessible from within the various sites.	solved
Inefficient_synchronization_on_specification_updates	Testbed adaptations made upon changes to the system specifications were easily communicated and synchronized across the involved teams.	solved
Non_comparable_test_results	The difference in test results across the various test and development teams was solved with respect to the testbed configuration divergence, given the established exchange and synchronization infrastructure and the utilized docker artefacts for testbed management.	solved
Inefficient_handling_of_defects	The time for handling and processing of tickets/defects by the development teams was largely accelerated given the increased reproducibility of results across the various teams and stakeholders.	solved
*Instabilities_in_*	The instabilities in the test scripts, emerging from the	

testbed_handling	divergent proprietary testbed configurations across various workstations, were intrinsically removed based on the proposed solution.	solved
Incompatible_cryptographic_material	The cryptographic material was unified within one centralized testbed instance that was collaboratively worked on across the various teams and partners.	solved
False_positives	The probability for a false positive result based on the divergent testbed configs and a misunderstanding of the technology or specification aspects was largely reduced provided the collaborative distributed approach based on gitlab *pull*, *push* and *merge* commands. Thereby, regular test and reviews of testbed changes were applied until proposed changes were approved and established across the involved teams as a basis for further testing.	solved

VI. Experimental Results

The current section focuses on the computational performance of our proposed solution in the course of increasing the robustness of the test execution process and correspondingly improving the failure analysis with respect to the SUT in question. We focused on this high level type of evaluation given that the belonging project is of industrial nature and should be discussed about only in an abstracted manner - i.e. no specific testbed code and configurations can be provided. At this point of the presentation, it should be remarked that the parameters of the host on which the presented measurements were conducted are briefly summarized in Table 3.

Table 3: Parameters of the Host utilized for the Measurements

Modell	ThinkPad T470 Signature Edition
Processor	Intel (R) Core (TM) i5-7200U CPU@2.50GHz 2.71 GHz
RAM	24,0 GB (23,9 GB usable)
System type	64 Bit Operating System X64-based Processor

In order to improve the stability in the course of regression testing for one of the access network components as a device under test, the rest of the unified testbed had to be regularly restarted, such that a defined network configuration is reset and the following test results can be interpreted in a clear and solid way. Indeed, a testbed restart was required after each single test case execution, which has drastically improved the test execution process in its stability and has led to better quality of the resulting defect tickets as well as improved collaborative failure analysis between the development and the product testing team.

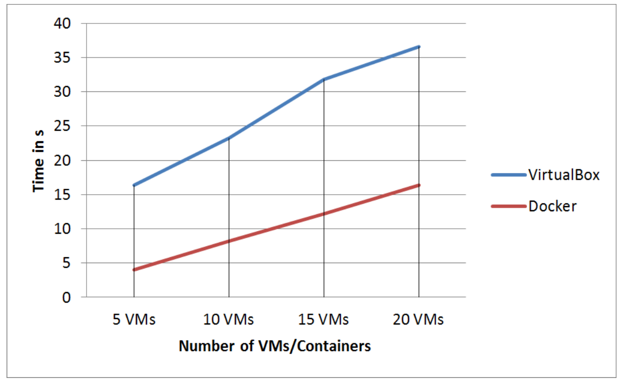

FIGURE 3: TIME COMPARISSON FOR THE CASE OF RESTARTING THE TESTBED WITH A VARYING NUMBER OF INVOLVED COMPONENTS

As discussed, the initial proprietary testbed solution was based on hypervisor technology and was spanning over a large number of virtual machines which took a long time to restart and setup a belonging defined network and testbed configuration state. Furthermore, due to the fact that every time the overall hypervisor system had to be restarted, which led to intense interactions with the underlying host, there were a number of instabilities on virtual hardware level, especially when it comes to the assignment and numeration (eth0, eth1 …) of network interfaces within the Linux testbed nodes. These instabilities have naturally led to problems on network management level such as wrong subnet numbering and IP address assignment.

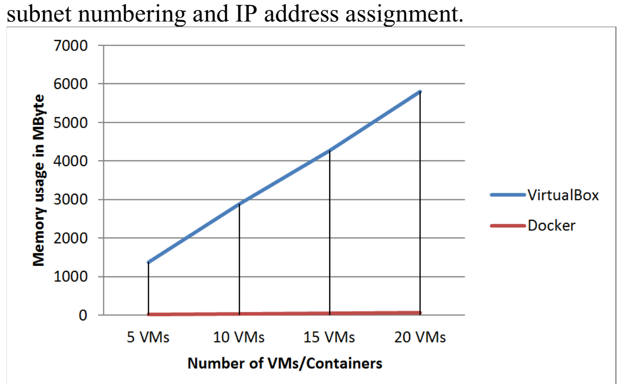

FIGURE 4: MEMORY UTILIZATION IN THE COURSE OF RESTARTING THE TESTBED WITH VARIOUS NUMBERS OF INVOLVED COMPONENTS

Generally, it can be summarized that the unified *docker* based testbed was much more stable than the proprietary VirtualBox solution, which was initially utilized and proprietarily modified by each involved party (product test, security test, load- and performance-test). Furthermore, it could be clearly observed that the container based solution was much faster in terms of restarting time for different numbers of nodes from the testbed as depicted in Figure 3. Thereby, the time measurements with respect to the time required for testbed restart is clearly in favor of the container based unified framework, which has led to increased test execution effectiveness and easier debugging of test cases and the SUT, in case of failed test cases and test steps.

In addition to the above aspects, Figure 4 and Figure 5 outline the memory consumption on the machine hosting the testbed as well as the CPU utilization on the host machine. Both figures clearly underline the increased effectiveness and low overhead of the unified testbed approach based on container technology and a common base image. Figure 5 contains a peak at the value 10 VMs, which is indeed difficult to explain solely based on the comparison between hypervisor and container technology. Our best explanation is attributed to potential processes (e.g. cron jobs) running in the background that must have skewed the monitored results. However, the overall trend of the observed values clearly indicates the achieved increased effectiveness as initially presumed. This increased effectiveness turned out to be a focal point within the project drastically improving the quality of the failure findings and enabling the goal oriented and efficient collaboration between the test and development teams.

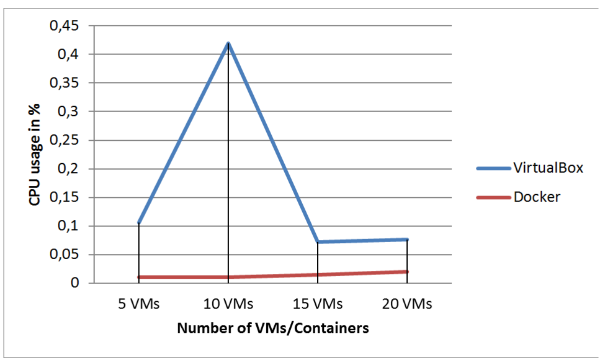

FIGURE 5: CPU UTILIZATION IN THE COURSE OF RESTARTING THE TESTBED WITH VARIOUS NUMBERS OF INVOLVED COMPONENTS

VII. CONCLUSIONS AND FUTURE WORK

The current paper presented on our experiences related to the need for a unified testbed management in a large scale integration project executed by a telecom service provider within the eHealth domain. Thereby, a significant number of independent parties and stakeholders were involved and adopted a hypervisor based solution for their own specific needs, e.g. in the scope of load- and performance testing, integration testing, penetration testing, security audits etc. Hence, the described situation led to a chaos, where different distributed testbed changes were not even announced on project level and reported defects and failed test cases were extremely hard to handle, given the lack of unified information regarding the testbed configuration in a highly complex network and services environment, involving a number of intertwined network and software stacks (e.g. DNSSEC, NTP, OCSP, HTTP proxies, IPSec …).

In order to remediate the above issues, we had to collaboratively work out a solution that would enable the continuous sharing of testbed configuration among different teams. Hence, given the conducted project analysis we implemented a solution based on container technology, i.e. *docker*, instead of the legacy hypervisor approach using VirtualBox or similar hypervisor settings. This approach included the involvement of various tools and frameworks such as *gitlab*, *docker-compose*, *docker-registries*, as opposed to other potential approaches based on *SVN* and *ssh-scripts* including tools such as *vagrant* and *Ansible*. The proposed solution enables the instant sharing of changes to the testbed configuration management and the transparency when it comes to tracing and identifying the root cause for a test case failure and belonging defects within the development teams. Hence, this enables the resolution of typical mistakes conducted within the initial project setup such as COMPLEX ENVIRONMENT and INEFFICIENT FAILURE ANALYSIS, as discussed in previous publications.

The efficiency of the proposed solution was further underlined by a series of experiments relating to the stability of the test execution procedure. Thereby, we measured the speed as well as the computational overhead within the underlying host, relating to the restart of a various number of involved testbed components within the test case execution process. These numerical measurements clearly show that the unified testbed management solution improves the overall test approach by a large magnitude thereby scaling up the (testbed configuration) sharing, the efficiency, the speed and reducing the overall computational overhead of the test process.

REFERENCES

[1] N. Tcholtchev, M. A. Schneider and I. Schieferdecker, "Systematic Analysis of Practical Issues in Test Automation for Communication Based Systems", 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Chicago, IL, 2016, pp. 250-256

[2] HP Quality Center: https://www.microfocus.com/en-us/products/quality-center-quality-management/overview, as of date 29.04.2019

[3] TestRail: https://www.gurock.com/testrail, as of date 29.04.2019

[4] Jira: https://www.atlassian.com/software/jira, as of date 29.04.2019

[5] 5G Playground of Fraunhofer FOKUS: https://www.fokus.fraunhofer.de/go/en/fokus_testbeds/5g_playground, as of date 29.04.2019

[6] 5G Berlin: https://www.5g-berlin.org/, as of date 29.04.2019

[7] F.Kaltenberger, R.Knopp, N.Nikaein, D.Nussbaum, L.Gauthier, C.Bonnet, "OpenAirInterface:Open-source Software Radio Solution for 5G", European Conference on Networks and Communications (EUCNC), Paris, France, July 2015.

[8] TAHI-TestSuite: https://www.ipv6ready.org.cn/home/views/default/resource/logo/phase2-core/index.htm, as of date 29.04.2019

[9] IPv6 Forum: http://www.ipv6forum.com/, as of date 29.04.2016
[10] J. Ruiz, A. Vallejo and J. Abella, "IPv6 conformance and interoperability testing", 10th IEEE Symposium on Computers and Communications (ISCC'05), Murcia, Spain, 2005, pp. 83-88. doi: 10.1109/ISCC.2005.87
[11] Vallejo, J. Ruiz, J. Abella, A. Zaballos and J. M. Selga, "State of the Art of IPv6 Conformance and Interoperability Testing", in IEEE Communications Magazine, vol. 45, no. 10, pp. 140-146, October 2007. doi: 10.1109/MCOM.2007.4342835
[12] ETSI (European Telecommunications Standards Institute): http://www.etsi.org/, as of date 29.04.2019
[13] gematik eHealth infrastructure: https://fachportal.gematik.de/, as of date 29.04.2019
[14] Peng Li, "Selecting and using virtualization solutions: our experiences with VMware and VirtualBox", Journal of Computing Sciences in Colleges archive, Volume 25 Issue 3, January 2010, Pages 11-17
[15] VirtualBox: https://www.virtualbox.org/, as of date 29.04.2019
[16] Qemu: https://www.qemu.org/, as of date 29.04.2019
[17] Linux KVM: https://www.linux-kvm.org/page/Main_Page, as of date 29.04.2019
[18] Vmware: https://www.vmware.com/, as of date 29.04.2019
[19] Carl A. Waldspurger, "Memory resource management in VMware ESX server", ACM SIGOPS Operating Systems Review - OSDI '02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation, Volume 36 Issue SI, Winter 2002 I, Pages 181-194
[20] Xen Project: https://xenproject.org/, as of date 29.04.2019
[21] docker: https://www.docker.com/, as of date 29.04
[22] Carl Boettiger, "An introduction to Docker for reproducible research", ACM SIGOPS Operating Systems Review - Special Issue on Repeatability and Sharing of Experimental Artifacts, Volume 49 Issue 1, January 2015, Pages 71-79
[23] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," in IEEE Cloud Computing, vol. 1, no. 3, pp. 81-84, Sept. 2014. doi: 10.1109/MCC.2014.5
[24] C. Anderson, "Docker [Software engineering]," in IEEE Software, vol. 32, no. 3, pp. 102-c3, May-June 2015. doi: 10.1109/MS.2015.62
[25] OpenShift: https://www.openshift.com/, as of date 29.04.2019
[26] OpenStack: https://www.openstack.org/, as of date 29.04.2019
[27] OpenNebula: https://opennebula.org/, as of date 29.04.2019
[28] Kamal Benzekki, Abdeslam El Fergougui, Abdelbaki Elbelrhiti Elalaoui, "Software-defined networking (SDN): A survey". Security and Communication Networks, 2016, 9 (18): 5803–5833. doi:10.1002/sec.1737.
[29] Masayoshi Kobayashi, Srini Seetharaman, Guru Parulkar, Guido Appenzeller, Joseph Little, Johan van Reijendam, Paul Weissmann, Nick McKeown, "Maturing of OpenFlow and Software-defined Networking through deployments", Computer Networks, Volume 61, 2014, Pages 151-175, ISSN 1389-1286, https://doi.org/10.1016/j.bjp.2013.10.011.
[30] Fraunhofer FOKUS IPv6 Test and Network Simulation Lab: https://www.fokus.fraunhofer.de/go/en_ipv6lab, as of date 29.04.2019
[31] Fraunhofer FOKUS Conformance and Interoperability Lab: https://www.fokus.fraunhofer.de/go/en_conformance_lab, as of date 29.04.2019
[32] Test Automation Patterns: https://testautomationpatterns.org/wiki/index.php/Main_Page, as of date 29.04.2019
[33] Standard for Software Test Documentation. Technical report, IEEE 829, 2008.
[34] RFC 2544: Benchmarking Methodology for Network Interconnect Devices, https://tools.ietf.org/html/rfc2544, as of date 29.04.2019
[35] L. Chen, "Microservices: Architecting for Continuous Delivery and DevOps," 2018 IEEE International Conference on Software Architecture (ICSA), Seattle, WA, 2018, pp. 39-397.doi: 10.1109/ICSA.2018.00013
[36] F. J. Meng, M. N. Wegman, J. M. Xu, X. Zhang, P.Chen and G. Chafle, "IT troubleshooting with drift analysis in the DevOps era," in IBM Journal of Research and Development, vol. 61, no. 1, pp. 6:62-6:73, 1 Jan.-Feb. 2017.doi: 10.1147/JRD.2016.2630478
[37] Continuous Testing: https://www.tricentis.com/products/what-is-continuous-testing/, as of date 10.06.2019