NESSEE: An In-House Test Platform for Large Scale Tests of Multimedia Applications Including Network Behavior

Robert Lübke, Daniel Schuster and Alexander Schill

Computer Networks Group
Technische Universität Dresden
Dresden, Germany
{robert.luebke,daniel.schuster,alexander.schill}@tu-dresden.de

Abstract. This paper presents the test platform NESSEE that can be used for reproducing network behavior in large-scale experiments with distributed systems. The main target group of our platform consists of companies that require an easy-to-use in-house emulation environment for testing their developed applications. In our scenario we use NESSEE to test the video conferencing software of our industry partner Citrix. We illustrate this with examples from scalability, functional and network testing. The evaluation of the concepts shows its applicability in other scenarios as well.

Key words: NESSEE, emulation platform, network characteristics, application behavior, testing

1 Introduction

During the development of an application, there should be a testing phase in which the application is executed under different real-world conditions. Various inputs are passed to the program to examine if it can meet its requirements and to find faulty behavior. In current software engineering models, this testing phase already starts in an early stage of the implementation using prototypes and is repeated regularly. This requires much coordination efforts, especially when testing large distributed systems. Therefore, there is the need of tool support and automation to facilitate this recurring process.

Every application communicating over a network must also be tested under various network conditions, especially if the overall system is distributed over several nodes. To do this you have to find a suitable environment to perform those tests in. Companies usually use the targeted *production environment* of the application for this. The results have great practical relevance, but often this approach is not feasible because of security reasons and confidentiality of the software. An alternative is a *dedicated test environment* with characteristics similar to the target environment. But rebuilding a global-scale infrastructure for large-scale experiments is time-consuming and expensive. *Simulation* is a synthetic approach, where the application itself and all necessary conditions, for

example the network behavior, are transformed into a model that is used for calculations. Due to model complexity it is only applicable in small scenarios. Furthermore, the abstraction of the original application under test may lead to less accurate tests and experiments. In the research community there are *federated test platforms* in which emulation can be used to reproduce network behavior. Emulation is a combined approach using the original application and calculating only some of the real-world conditions with a model. Unfortunately, these federated systems are missing means for automated testing and are not suitable for most companies.

We therefore developed own concepts for an in-house emulation platform for large-scale tests with distributed systems and implemented these in a testbed called NESSEE (Network Endpoint Server Scenario Emulation Environment). In conclusion, our main contributions are:

- combination of a large-scale testbed with the fine-grained emulation of multiple network characteristics
- emulation of application behavior
- means to continuously integrate into the application developer's work flow
- a generic specification of test cases using a scripting approach

The remainder of this paper is structured as follows. At first, we discuss the problems of performing large-scale tests with real-time multimedia applications and derive certain requirements. We then highlight similar approaches and discuss their shortcomings. Then we present the main concepts of our own approach, the NESSEE platform, and illustrate some of the experiments it allows to perform. After the evaluation of our concepts, we finally conclude the paper.

2 Problem discussion and requirements

In our use case we require a test platform for performing various experiments and different kinds of tests with the video conferencing solution of our industry partner Citrix. Especially those real-time multimedia applications which are offered as software as a service have to reach a certain quality level to meet the users' expectations. This requires an extensive testing phase including the reproduction of occurring real-world effects that could influence the application.

One major aspect that must not be concealed is the network with all its characteristics and technology-specific effects, including basic network parameters like bandwidth limitation, packet delay and loss as well as advanced network parameters and effects like packet reordering, duplication, jitter, connection hand overs and disconnects. In the real world, the systems running the application are connected with each other via multiple hops and the network behavior depends on concurrent traffic. Therefore, emulation of network topologies of any complexity and background traffic are required. All mentioned parameters and effects should be re-configurable during run time of the experiment to increase the flexibility. The dynamic reconfiguration should be based on time and user interaction.

The coordination effort for the testers should be as small as possible, especially the creation of test cases should be facilitated. Furthermore, the aim is to run automated tests, which require the emulation of the application behavior. When performing scalability tests, thousands of software under test (SUT) instances need to run on the test systems and they must be started and controlled in an automated way. To achieve this, the SUT must provide some kind of control interface that can be used by the emulation platform. The necessary adaptions of the SUT for this interface should be as slight as possible to emulate the real application behavior as precise as possible. Another way to reduce user efforts is to support current best practices of software engineering like continuous integration and test-driven development. The emulation platform should be able to seamlessly integrate into the application developer's work flow.

To be able to share test resources among users we require a multitenancy approach, allowing multiple users to run experiments concurrently. Of course, the emulation platform is supposed to handle multiple large experiments at once. This especially requires a good scalability of the overall system.

3 Related work

In the last decade much research work has been done in the area of large testbeds for realistic measurements and experiments. FIRE [1] and GENI [3] are initiatives supporting the development of those testbeds. Some existing and well-established testbeds, each designed with different goals, are presented in the following.

OneLab [2] and PlanetLab [6] are very large and well-known testbeds supporting the development of new network services. PlanetLab provides more than thousand nodes all over the world that can be used for experiments. However, the number of users is also quite big which makes it difficult to reserve enough resources for own experiments. G-Lab [10] is an experimentation platform for future Internet studies and development with the main focus on routing, mobility and security. The G-Lab testing facilities are located in several German universities and consist of wired and wireless hardware with over 170 nodes. The Topology Management Tool [9] allows the user to create virtual network topologies using the G-Lab experimental facility. Emulab [11] is a network testbed offering various different experimental environments, for example emulation, simulation as well as live experimentation in the Internet. Resources like hosts, routers and networks are virtualized and the experimenter just specifies the desired topology and characteristics. The experiment itself is then launched on selected suitable machines and switches. Emulab is open source software and was used to build many independent testbeds all over the globe. ProtoGENI [8] extends Emulab and follows the approach of bridging different physical sites into one testbed. ORBIT [7] focuses on wireless network experiments. The experimental facilities consist of about 400 wireless nodes and support both, network emulation and field-trial capabilities. FEDERICA [4] is a European testbed very similar to GENI, using the concepts of sliced infrastructure for processing and networking.

Despite efforts of the providers the presented federated research testbeds do not completely fit the needs of companies, which prefer in-house solutions because of legal and security reasons. Some testbeds have complex registration procedures and unflexible membership dues for companies. Furthermore, all mentioned testbeds focus on research instead of software testing. They provide the means for manually performing non-recurring experiments. If the results of the experiment are successfully obtained, it usually gets archived. Software testing on the other hand requires recurring and reproducible test runs as well as means for automation. Beside the detailed reproduction of network conditions, automated software testing also requires the emulation of application behavior during the experiment. We therefore developed an own test platform that is chiefly based on well-established concepts of existing network testbeds, but that is also able to meet the requirements of testing software, especially multimedia applications.

4 Concepts of the NESSEE test platform

This section covers the main concepts of our own test platform called NESSEE. Its general architecture is illustrated in Figure 1. Client/server-based distributed systems and especially multimedia applications are the main focus of the platform. Therefore the environment consists of **Test Systems** for emulating the client- and server-side **Software Under Test** (SUT) behavior. As discussed previously companies usually prefer in-house solutions. Therefore the test systems are either physical or virtual machines inside the company network. If security is not that relevant, the test systems could of course be located at cloud computing providers. NESSEE integrates with Amazon EC2 allowing to start up and shut down virtual test systems dynamically. To achieve the large number of applications for scalability tests, multiple SUT instances must be executed on a test system. A performance metric is used to determine how many instances can be handled. Each test system further runs the **TestNodeModule** (TNM), which starts and controls the SUT instances via **Inter Process Communication** (IPC). Depending on the used operating system the supported IPC mechanisms are *COM* (Windows), *Unix command-line access*, *SSH* and a *custom IPC mechanism* using message passing. The SUT developer just has to implement one of these mechanisms to make the SUT controllable via NESSEE. As this usually requires only slight adaptations, it enables NESSEE to emulate the real application as accurately as possible without abstracting from the original SUT.

The central **NESSEE Server** coordinates all components involved, including the test systems running the TNM. It further collects log files from the SUT and NESSEE components and stores them in the central log repository for later inspection by the tester. A multi-tenant web front end can be used to control and configure the NESSEE Server. Additionally, one can use a Web service API to interact with NESSEE. This is especially useful for build and continuous integration software that automatically trigger NESSEE tests. Using this approach, NESSEE seamlessly integrates into the development process of the SUT.

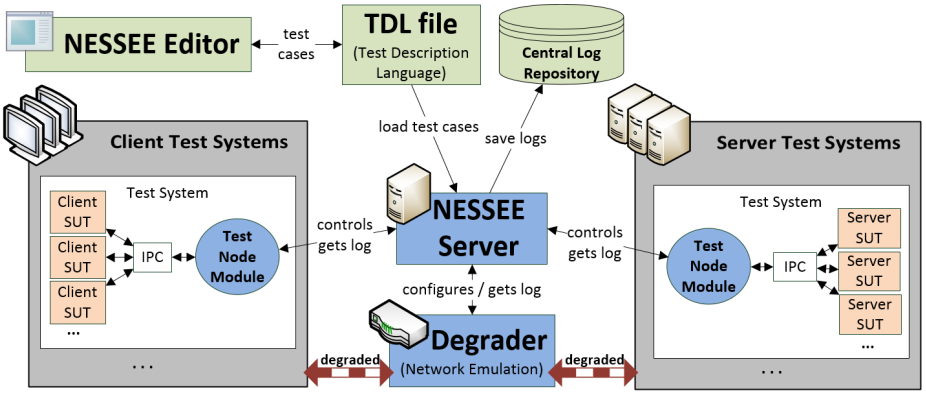

Fig. 1. General architecture of the NESSEE platform

The actual test cases are described in a dedicated format using the generic XML-based **Test Description Language** (TDL). It consists of modules describing the network conditions, the application behavior and the composition of the test case. Additionally, the testers can use *JavaScript* to specify the application behavior within scripts. During the test, these scripts are executed by script engines inside the NESSEE Server and the TNM. The scripting approach provides the tester with all means of a programming language and therefore allows a flexible specification of the application behavior. The **NESSEE Editor** is a graphical environment for authoring TDL modules as well as scripts and thus reduces the efforts of test case creation.

The **Degrader** component is responsible for the emulation of the desired network behavior. It acts as a router for all test systems and artificially deteriorates the characteristics of all incoming traffic based on source IP address and port. Before a test run is started, the Degrader is configured by the NESSEE Server according to the network description of the test case. Among others, the Degrader emulates complex topologies, bandwidth limitations, packet delay, jitter, loss, reordering and duplication as well as connection handovers and disconnects. Detailed information about NESSEE's network emulation is given in [5].

5 Types of experiments

This section discusses common types of experiments that are necessary when testing multimedia applications. We further illustrate and give concrete examples of how these experiments can be performed with the help of our previously discussed concepts.

5.1 Functional testing

Functional testing is essential not only in the multimedia scenario, but for all kinds of software. The functions of the software are tested by providing input and comparing the output with the expectations that come from the software specification. At first, the functions of the SUT must be identified. As we use well-defined interfaces to the SUT (see Section 4) this list of functions is given by the SUT's API. The creator of the test can then write a script that calls the corresponding methods of the SUT via its helper object. When creating the input, especially borderline cases should be checked. The expected output is also specified in the script and is compared to the actual output during the test run. This process is illustrated in Figure 2. In this example, the SUT just provides two functions, one for setting a desired video resolution and one for getting it. If setting fails or getting it afterwards does not return the expected resolution, the test fails.

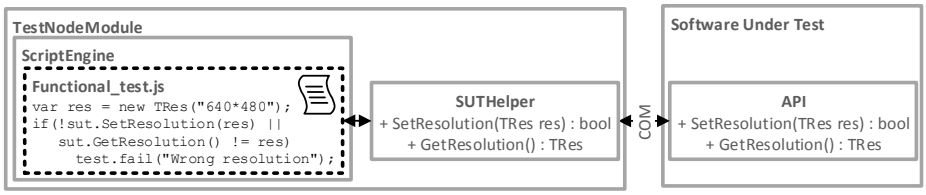

Fig. 2. Example of a functional test in the NESSEE platform

5.2 Scalability testing

Scalability testing is a type of non-functional testing and its aim is to test whether or not a system behaves as expected when it is changed in size in order to meet a growing need. In multimedia applications, scalability plays an important role for server-side components, that serve a certain number of clients. The objective is to find out the maximum number of clients and to examine the behavior under heavy load.

With the presented concepts, NESSEE is able to handle those scalability tests with minimal coordination effort for the testers. Such large-scale tests are described in TDL like any other test and behavior of the applications can be scripted. The emulation platform automatically chooses suitable test systems, deploys and starts the SUT and the tester can concentrate on the test itself even if many test systems and SUT instances are involved in the test.

In our use case, we examine the scalability limitations of the video conferencing server components running experiments with up to 6,000 client SUT instances. Figure 3 is a screen shot of the web front end, giving an overview about involved SUT instances running on the test systems.

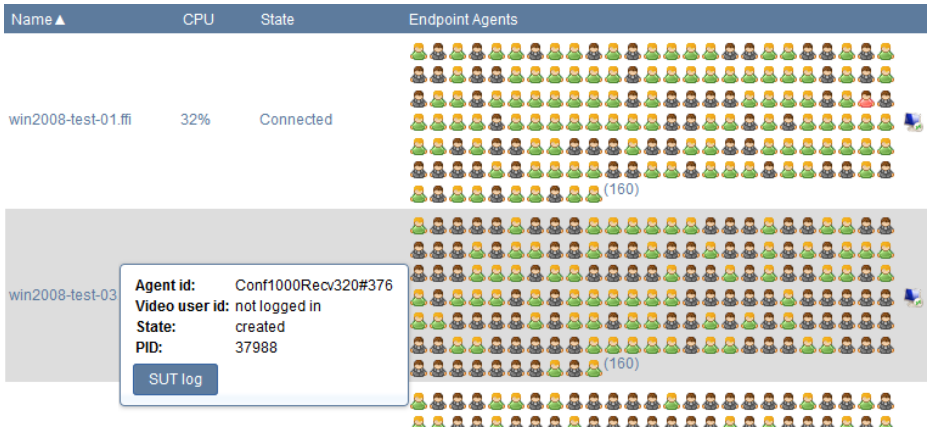

Fig. 3. In large-scale tests the web front end visualizes the states of the SUT instances with different icons and provides detailed information via tool tips.

5.3 Network testing

The aim of network testing is to examine the behavior of the software under various network conditions. Basically, every application communicating over a network should undergo those tests, but it is essential for real-time multimedia applications, because the quality and the user experience directly relate to the network conditions. As discussed in Section 4, the NESSEE emulation platform supports the fine-grained configuration of a variety of network parameters. But this configuration is not static as network testing requires the network conditions to change. Therefore, the testers have the possibility to reconfigure the characteristics manually using the web front end, but also automatically using the test scripts. Figure 4 illustrates this with an example of a network test. Changing the network characteristics is usually combined with one of the previously mentioned test types, functional or scalability testing.

6 Evaluation

NESSEE fulfills the analyzed functional requirements of a platform for automated tests of multimedia applications. In the evaluation of our concepts the non-functional requirement of scalability should be examined.

The **scalability and accuracy of the network emulation** components was already shown in previous work [5]. We evaluated the deviations of configured and actually measured values for various network characteristics in large and complex scenarios and produced the following results:

- data rate: ±3.50%
- packet delay: ±0.20%
- packet loss: ±0.10%
- packet reordering: ±0.13%
- packet duplication: ±0.00%

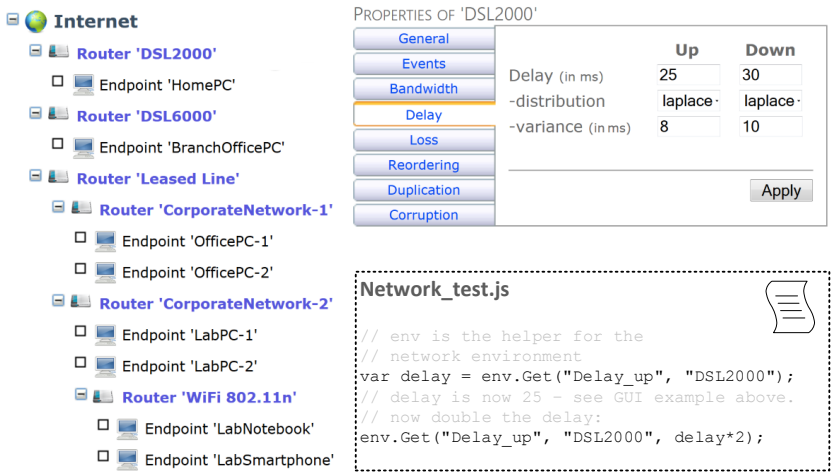

Fig. 4. For network tests the characteristics can be changed manually via the web front end and automatically via the test scripts.

All parameters are emulated accurately, except for the data rate, which still needs improvement. We also observed effects like bandwidth sharing and the variation of delay (jitter).

The **scalability of the script engine approach** was evaluated in multiple scenarios. We compared the NESSEE Script Engine with other similar engines regarding the performance of long-running functions, big data handling and multiple calls of smaller functions. Our engine did not always achieve best results, but it is sufficient for the usage inside the emulation environment. One quite important aspect is calling native code from JavaScript as illustrated in Figure 2 with the *sut* helper object that is resolved to the native class *SUTHelper*. This feature is used frequently by test scripts during the test run and therefore requires good performance. To evaluate this, we measured the execution time for a varying number of calls and compared it with other commonly used script engines. The results in Figure 5 show the very good performance of our script engine in comparison with other implementations.

We further investigated on the **scalability of the IPC approach** used to start and control the SUT. Regarding this, it is necessary to know how many SUT instances can be run on one machine at the same time. In our evaluation we used the COM interface that the video conferencing client provides and measured the starting time of a varying number of instances. The results of two different test machines are shown in Figure 6. Obviously, the results depend on the hardware performance. They show that it is possible to start over 500 instances of the SUT at the same time, although the start up phase was up to two minutes. Nonetheless, this amount of time is acceptable in such large-scale scenarios.

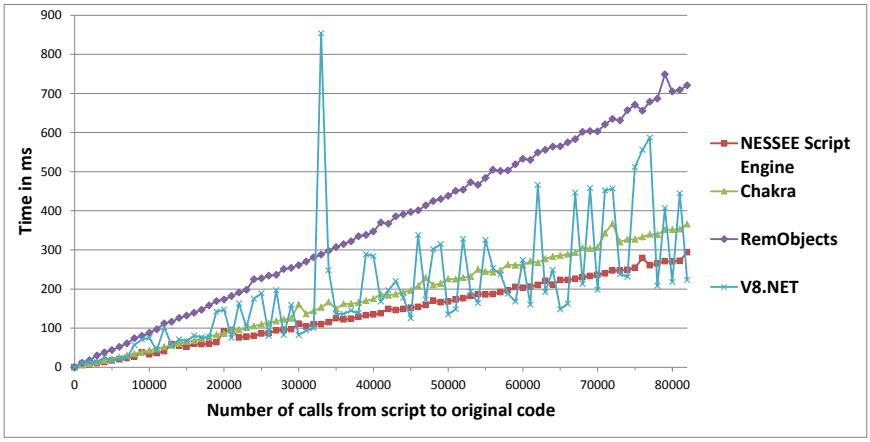

Fig. 5. Time for making calls from script to original code using different script engines

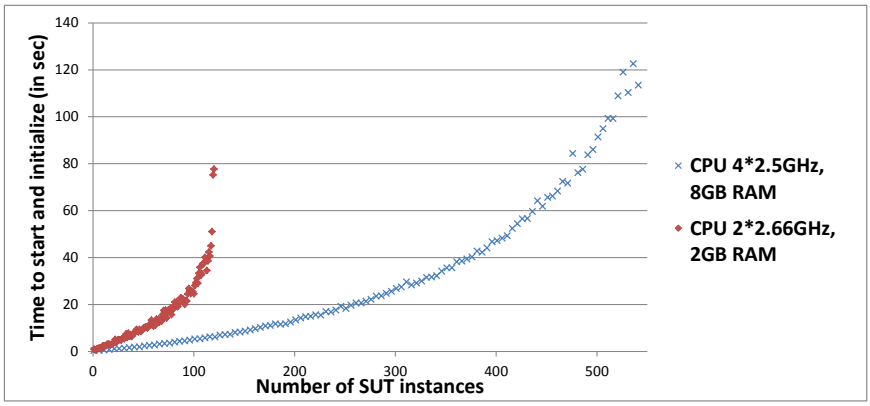

Fig. 6. Time to start a given number of SUT instances for two test systems

Beside these components, the **overall scalability** of the whole emulation platform was already shown in Section 5.2 with the illustration of the large-scale tests we performed with NESSEE.

7 Conclusion

This paper presented concepts of a platform for various kinds of tests with multimedia applications. Most of the concepts originate from existing testbeds for network research and were adapted to the use case of automated software testing. Especially the in-house approach makes the platform attractive for companies. We implemented and evaluated our concepts in the NESSEE platform. Our main contribution is the combination of accurately emulating various network

characteristics as well as application behavior inside a large-scale test platform. Furthermore, we provide means to continuously integrate testbed experiments in the software development workflow. Finally, we created a generic specification of test cases using a description language and a scripting approach. Although the focus of this work is testing of multimedia applications, the generic concepts can easily be adapted to tests of any large-scale distributed system.

Acknowledgment. This work results from the NESSEE research project that has been financed by Citrix Systems, Online Services Division. The authors would like to thank the project members for their contributing work.

References

1. "FIRE," http://www.ict-fire.eu, accessed: 03/24/2014.
2. "Onelab. future internet testbeds," http://www.onelab.eu, accessed: 03/24/2014.
3. M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "Geni: A federated testbed for innovative network experiments," *The International Journal of Computer and Telecommunications Networking*, 2014.
4. M. Campanella and F. Farina, "The federica infrastructure and experience," *The International Journal of Computer and Telecommunications Networking*, 2014.
5. R. Lübke, D. Schuster, and A. Schill, "Large-scale tests of distributed systems with integrated emulation of advanced network behavior," *IADIS International Journal on WWW/Internet*, vol. 10, no. 2, 2012.
6. L. Peterson and T. Roscoe, "The design principles of planetlab," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 11–16, Jan. 2006. [Online]. Available: http://doi.acm.org/10.1145/1113361.1113367
7. D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols," in *Wireless Communications and Networking Conference, 2005 IEEE*, vol. 3. IEEE, 2005, pp. 1664–1669.
8. R. Ricci, J. Duerig, L. Stoller, G. Wong, S. Chikkulapelly, and W. Seok, "Designing a federated testbed as a distributed system," in *Testbeds and Research Infrastructure. Development of Networks and Communities*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, T. Korakis, M. Zink, and M. Ott, Eds. Springer Berlin Heidelberg, 2012, vol. 44, pp. 321–337. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35576-9_26
9. D. Schwerdel, D. Hock, D. Günther, B. Reuther, P. Müller, and P. Tran-Gia, "ToMaTo - a network experimentation tool," in *7th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom 2011)*, Shanghai, China, Apr. 2011.
10. D. Schwerdel, B. Reuther, T. Zinner, P. Müller, and P. Tran-Gia, "Future internet research and experimentation: The g-lab approach," *The International Journal of Computer and Telecommunications Networking*, 2014.
11. B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 255–270, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844152