IADIS International Journal on WWW/Internet
Vol. 10, No. 2, pp. 138-151
ISSN: 1645-7641

LARGE-SCALE TESTS OF DISTRIBUTED SYSTEMS WITH INTEGRATED EMULATION OF ADVANCED NETWORK BEHAVIOR

Robert Lübke. *Computer Networks Group, Technische Universität Dresden, Germany.*

Robin Lungwitz. *Computer Networks Group, Technische Universität Dresden, Germany.*

Daniel Schuster. *Computer Networks Group, Technische Universität Dresden, Germany.*

Alexander Schill. *Computer Networks Group, Technische Universität Dresden, Germany.*

ABSTRACT

Reproducing the characteristics of complex network infrastructures is not supported by current testing environments for distributed systems. We argue that the emulation of multi-level topologies and fine-grained network behavior is necessary to provide an adequate testbed for experiments with distributed systems. Therefore, we developed an extensible emulation environment called NESSEE, which emulates application behavior as well as network aspects. Evaluation results show its applicability for large-scale tests. Furthermore, the developed NESSEE platform is actually used by our industry partner Citrix Online for testing video conferencing and eCollaboration systems.

KEYWORDS

Emulation, network, topology, behavior, NESSEE, Degrader.

1. INTRODUCTION

Reproducing network characteristics is a well-established mechanism for example in network protocol development. It allows researchers to test new protocols or modifications of existing ones in a reproducible way. However, it can also be applied in network planning to predict consequences of certain infrastructure changes. Besides these use cases reproducing network characteristics is appropriate and even necessary for testing distributed systems. These tests often take place in laboratory environments with almost perfectly connected network nodes. In reality the parts of the distributed systems are located all over the globe and have to interact with each other overcoming thousands of kilometers and a significant number of network hops. During this communication packets get lost, delayed, reordered and corrupted. Concealing this behavior while testing distributed systems does not reflect the real world accurately enough.

Reproducing network conditions can be achieved by either simulation or emulation. While network simulation is a synthetic approach in which the whole networked system is mapped to a model that allows various calculations, network emulation is the imitation of a slow and unreliable network on a real physical network with better characteristics.

This work focusses on network emulation, because real network traffic must be exchanged between the nodes of a distributed system. A simulation of the application behavior would cause inordinate abstraction, because the software under test has to be reduced to a black box with certain distributions of incoming and outgoing packets.

In the following, we will present the basic principles of networks with relevant parameters and effects and identify the requirements of a test environment for distributed systems. After the discussion of related work, we present the concept of an emulation environment for scalability tests of distributed systems that includes network emulation. We further evaluate our approach, conclude the paper and point out the main objectives of our future work.

2. NETWORK PARAMETERS AND EFFECTS

The links and nodes of a computer network have different characteristics that depend on the used technologies. These characteristics can be specified with the help of various metrics. Furthermore, technology-specific effects can occur in the network. This section covers the most important network metrics and effects of typical computer networks. We focus on parameters and effects that can be observed at the IP layer and higher because network emulation is usually done at this layer.

One of the most important parameters is the data rate which is often called bandwidth. Here one has to differentiate between the capacity C, the available bandwidth A and the Bulk Transfer Capacity (BTC). The capacity C of a link is defined as the maximum possible IP layer data rate. If one link is used by multiple communications at the same time, the capacity is shared. This is often called bandwidth sharing. The available bandwidth A of a link is the unutilized fraction of the capacity over a certain time interval. BTC is the maximum throughput obtainable by a single TCP connection [Prasad et al, 2003]. Most Internet service providers shape the traffic of their customers to an artificial maximum data rate. These rates may vary upon time.

Another very basic parameter is the delay between sending and receiving a packet. The overall delay is the accumulated value of propagation delay D^P (through signal transmission), queuing delay D^Q (dependent on the current load of the routers) and transmission delay D^T (time to transmit all bits of one packet). As it is not trivial to determine the One Way Delay (OWD) in only one direction, mostly the Round Trip Delay (RTD) is specified. The RTD can be determined by simply measuring the time between sending a packet and receiving the corresponding answer. The OWD is usually not a constant, but it varies according to a Laplace distribution. This delay variation OWD is known as jitter on the application layer.

The loss of single or even multiple packets can occur because of overloaded routers or defective hardware / software. The loss ratio specifies the amount of packets that get lost during a communication relatively to the total amount of packets. This parameter is specified as One Way Loss Ratio (OWLR) or Round Trip Loss Ratio (RTLR) dependent on the reference. During the transmission packets can get corrupted, if some of the contained bits get flipped. However, those failures are covered and masked by lower layers and affect the IP layer with a higher loss ratio.

Packet duplications can occur because of unnecessary retransmissions or defective hardware/software. In such cases a packet arrives twice or even more often at the receiver. The Duplication Ratio (DR) is the amount of duplicated packets relatively to the total amount of packets.

If different packets of one communication can take different routes, it is possible that they arrive at the receiver in another order as they were sent. This effect is specified as the Packet Reordering Ratio (PRR).

Especially in wireless connections further effects like permanent or short-time disconnections can be observed. When users move while they are connected to mobile cellular networks handovers between the cells and between different technologies can occur.

3. REQUIREMENTS

When performing scalability tests of the video conference systems of our industry partner Citrix Online, we spotted some special emulation requirements for general testing of distributed systems. The support of complex network topologies is essential to allow emulation of networks at any complexity level. Furthermore, the emulator has to differentiate between up and down links. It should also allow dynamic configuration changes during the runtime of a test. The essential parameters for the network emulation are data rate, delay and loss. The other parameters and effects that are described in the previous section should also be emulated.

To achieve the necessary degree of realistic behavior it is important to use the original software under test and not an abstract model. Finally, it is required to control the large number of software under test instances during test runtime to emulate application behavior as well.

4. RELATED WORK

Much work has been done in the area of network emulation during the last two decades. The aim of this section is to give a brief overview on the state of the art of network emulation.

Emulation environments like *PlanetLab* [Peterson et al, 2006], *OneLab* [OneLab, 2012] and *EmuLab* [White et al, 2002] are a federation of multiple test networks of different scientific institutions and companies. Usually researchers can do experiments with these networks after they participated in the project by adding nodes to the test infrastructure. These environments have been enhanced by network emulation tools as described in [Carbone and Rizzo, 2001].

In [Schwerdel et al, 2011] a network emulation tool for the test environment G-Lab [Schwerdel et al, 2010] was developed. This tool called ToMaTo allows the definition of complex topologies and certain link characteristics with a graphical editor. Emulation environments fulfill most of the stated network-related requirements for large-scale testing of distributed systems (see Section 1). However, they are not applicable for this use case, because they are not flexible enough in terms of dynamic changes during the runtime of an experiment. Furthermore, the main focus of these environments is performing experiments for network development and not large-scale testing of distributed systems. Therefore, there are no means to control and emulate the application behavior.

When searching for other solutions to emulate network behavior we found many hardware based products of various manufacturers. These hardware emulators are able to shape traffic very precisely, but they do not support complex network topologies and have limitations concerning dynamic changes during the test run. Therefore, we focused our search on freely available software emulators due to their greater flexibility and adaptability. The most important emulators we found and examined are discussed in the following.

NISTNet [Carson and Santay, 2003] is able to emulate all of the effects listed above. However, NISTNet is not able to emulate network topologies. In addition it is not further developed and its beta level code (according to the official web site) prevents the application.

In contrast Netem/TC [Hemminger, 2005] was introduced as traffic shaper in most common Linux distributions since version 2.6. The provided functionalities to modify bandwidth, delay, loss and other effects based on various connection parameters like IP address or port number qualify Netem to work as a network emulation node inside a network emulation environment. Based on Netem the wide area network emulator WANem [Kalita and Nambiar, 2011] was developed. It provides extensions like an analyzer for measuring real network links and an option to emulate disconnects [Nambiar and Kalita, 2011]. However, it is not possible to use Netem or WANem for representing a whole network topology.

In FreeBSD 2.2.8 Dummynet [Carbone and Rizzo, 2010] became the standard traffic shaping component of the ipfw firewall. Its basic concept called pipes is pretty handy to create virtual network topologies. Although Dummynet is only able to emulate bandwidth, loss and delay, there are many extensions based on this software. KauNet [Garcia et al, 2007] adds the ability to configure bit errors and it introduces a pattern-based approach. By using time- or data-driven patterns the precision and reproducibility of Dummynet was enhanced [Garcia et al, 2008]. ModelNet [Vahdat et al, 2002] in contrast to KauNet uses an unmodified version of Dummynet to emulate huge topologies on a predefined number of computers.

Recent work by [Nussbaum and Richard, 2009] showed that the emulation quality of the emulators Dummynet, NISTNet and Netem/TC is reasonable, although some emulators are still facing problems in certain scenarios. Dummynet for example still has problems with the accurate emulation of high data rates.

This section showed that there are already some solutions that can emulate network topologies and link characteristics as we require it for large-scale testing of distributed systems. But these solutions focus on network development experiments and therefore do not provide any means to integrate the emulation of the network behavior with the emulation of the application behavior. To our knowledge there is no system combining the emulation of network and application behavior in an integrated way to facilitate testing large distributed systems.

From all the mentioned network emulators, Dummynet with its extension KauNet seems to be the best tool to emulate network topologies of any size and complexity. Therefore, we use it as a basis for our integrated emulation platform NESSEE that is presented in the following section.

5. THE NESSEE PLATFORM

NESSEE (Network Endpoint Server Scenario Emulation Environment) provides a generic architecture for scalability tests of client/server-based distributed systems including network emulation. The general architecture and its most important components are presented in the following.

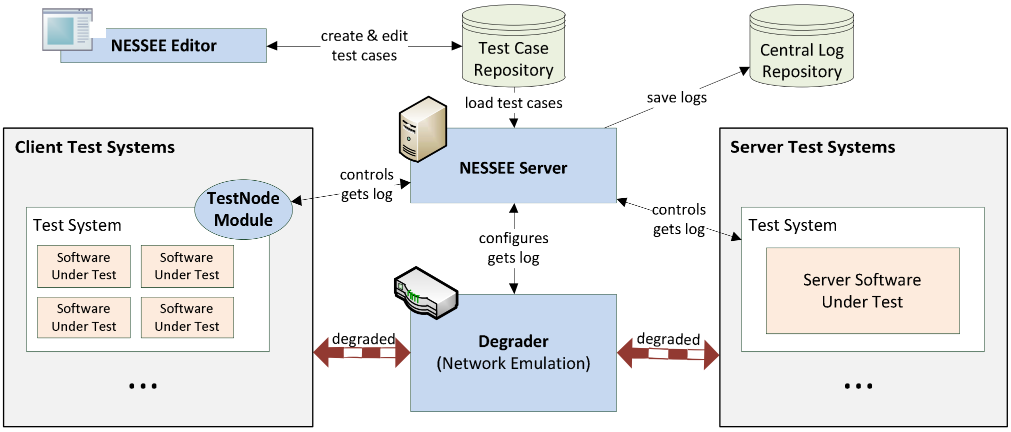

Figure 1. General architecture of the NESSEE platform

5.1 General Architecture

As NESSEE is an emulation environment for typical client/server-based systems, the **Software Under Test** (SUT) can be the client software as well as the server software. The SUT is installed on various test systems that are usually virtual machines (VM), but physical machines can also be used. Typically, multiple instances of the client SUT run on one **Client Test System**. **Server Test Systems** usually contain only one instance of the server SUT.

Both types of SUT (client and server software) communicate with each other. All the communication is routed through the **Degrader**, as it is the standard gateway of all test systems. The Degrader is responsible for the network emulation. The network conditions of each SUT instance can be configured separately based on IP addresses and ports, which allows the differentiation of multiple SUT instances on one machine.

The central component **NESSEE Server** manages the test systems and coordinates the SUT and the Degrader according to a certain **test case**. The NESSEE Server also has a web interface, which the testers can use to manage their test systems, create and edit tests, control running tests and to analyze finished tests.

The client SUT instances are controlled via a **Test Node Module** (TNM), which also runs on each client test system as it follows an agent-based approach. Besides controlling the applications, the TNM agents help fetching logs and statistics.

Additionally, the test cases contain JavaScript based test scripts that are executed by the NESSEE Server and TNM during the test. These test scripts allow controlling the test execution in a very flexible way. This enables the emulation of the SUT's behavior, which we found to be very important in addition to the emulation of network behavior. The SUT instances are not informed directly about changing network conditions as this would also not be done in reality.

The results of each test, its statistics and logs are stored in the **Central Log Repository**. The NESSEE Server retrieves the test cases from the **Test Case Repository**. These test cases are described in a generic, XML-based **Test Description Language** (TDL, see Section 3.2). The **NESSEE Editor** is a TDL authoring tool that enables the graphical modeling and detailed specification of test scenario descriptions within an independent application.

The main components of the NESSEE environment for application and network emulation are shown in Figure 1. In the following, the most important aspects of network emulation are discussed in detail.

5.2 Test Description Language

The Test Description Language is an XML-based generic description language that is used to formalize test cases that can be executed by the NESSEE Server.

The design of the TDL follows a modular concept and the modules are as independent from each other as possible. This allows the reuse and combination of components into a test case description that is executed on the test systems in the end. The **NetworkTopology** module defines the different network parameter sets and concrete network topologies. The **Behavior** module allows the definition of actions that can be arranged in complex flows. The **TestCaseDescription** links to the other modules and arranges the elements that are defined there into single test case representations.

In this work we focus on the network emulation and therefore provide an example of the NetworkTopology module. Here the tester can define different sets of network parameters called `NetworkCapabilities`. Listing 1 provides an example of the capability definition including bandwidth (kbps), delay (ms) and its variation (ms), loss (percentage), reordering (percentage), duplication (percentage) and a disconnection event every 24 hours. Every single parameter can have a direction attribute that indicates whether it should be used for uplink, downlink or both. To define the client-side network topology we implicitly use the XML structure. The child elements of a `NetworkNode` (gateway, router) can be sub network nodes and `NetworkEndpoint`s (the actual user computer). This enables modeling network topologies of any complexity. Every network node and endpoint can reference one of the previously defined capability parameter sets by its id. The server components of the distributed systems (`ServerCluster` in Listing 1) are modeled to run in `DataCenter`s that are connected to the Internet via multiple Internet service providers (ISP). Each `ISP` can also have a special set of predefined network capabilities.

```
<NetworkCapabilities id="DSL">
    <delay direction="both" variation="7"
            distribution="laplace">12</delay>
    <loss direction="up">0.1</loss>
    <loss direction="down">0.09</loss>
    <reordering direction="both">0.3</reordering>
    <duplication direction="both">0.001</duplication>
    <disconnect start="24h" duration="5s" />
</NetworkCapabilities>
<NetworkCapabilities id="DSL6000" BasedOn="DSL">
    <bandwidth direction="down">6000</bandwidth>
    <bandwidth direction="up">1000</bandwidth>
</NetworkCapabilities>
<!-- ... -->
<ClientTopology>
    <NetworkNode id="DSLRouter" NetworkCapabilitiesId="DSL6000">
        <NetworkEndpoint id="UserPC" />
        <NetworkNode NetworkCapabilitiesId="WiFiRouter">
            <NetworkEndpoint id="UserNotebook"
                    NetworkCapabilitiesId="WiFiEndpoint"/>
        </NetworkNode>
    </NetworkNode>
</ClientTopology>
<ServerComponents>
    <Datacenter id="datacenter_europe">
        <ISP id="isp01" NetworkCapabilitiesId="ISPCaps01"/>
        <ISP id="isp02" NetworkCapabilitiesId="ISPCaps02"/>
        <ServerCluster id="cluster01" defaultISPId="isp01"/>
    </Datacenter>
</ServerComponents>
```

Listing 1. Example definition of network capabilities, a simple topology and one server component

5.3 Degrader

The network emulation component Degrader is realized as a dedicated machine due to performance issues and it operates on the network layer. We used Dummynet as a basis because it meets most of the important requirements and because of its extensibility. Dummynet is used for the emulation of the basic network parameters. Its extension KauNet also allows us to emulate advanced network parameters and effects. Due to our modular design the testers can choose which one should be used. Both approaches are presented in the following.

5.3.1 Emulation of the Basic Network Parameters

The NESSEE Server provides a module called Degrader Control which binds the Degrader machine to the server. After parsing the TDL NetworkTopology module (see Listing 1) into a NESSEE internal topology model, the creation of the Dummynet configuration is done within an own data structure called **Binding Model**. Figure 2 shows that this model has the same structure as the NESSEE internal topology model. Its nodes are bound to equivalent nodes of the topology model. Two additional objects representing a **Dummynet pipe** and a **Dummynet rule** are attached to each node. A Dummynet pipe represents a channel with certain characteristics. These characteristics are derived from the network capabilities configured in the test case description. A Dummynet rule basically works like a firewall rule. It determines which packets have to pass which pipes based on IP addresses and port numbers of its source and destination.

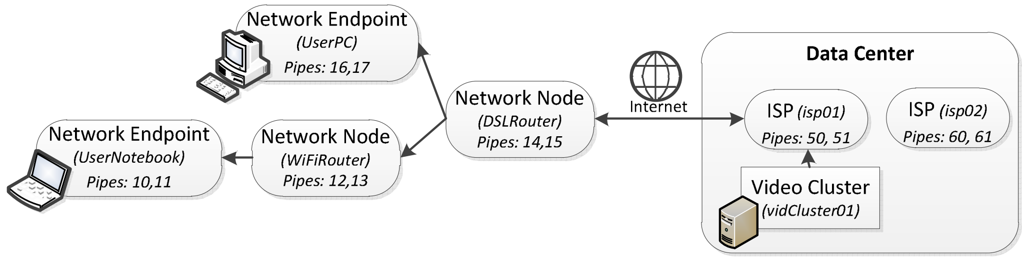

Figure 2. The Binding Model for the topology definition in Listing 1

Before starting a test case the Binding Model with all its nodes, pipes and rules is converted into the actual configuration commands of the Dummynet emulator which are then sent to the Degrader. Figure 3 illustrates the physical data flow from the client to the server test systems via the Degrader. All incoming traffic is forwarded to the corresponding pipe hierarchy.

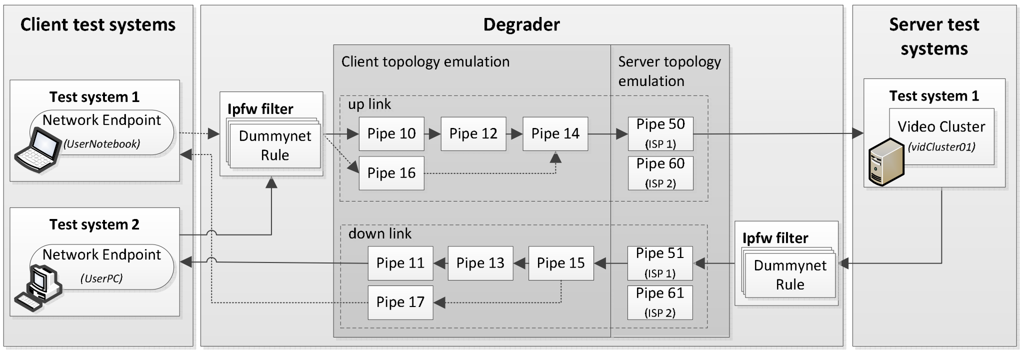

Figure 3. The physical data flow from the client test systems to the server test systems (up link) and back (down link) as well as its emulation inside the Degrader (example corresponds to Figure 2)

Both, Dummynet pipes and Dummynet rules of the Binding Model use an event-based approach to react on dynamic changes of the topology model, for example initiated by the tester or the test execution engine. This enables the topology model equivalent structure of the

Binding Model to easily realize dynamic changes during runtime of a test. Such changes are handled by the data structure and sent to the Degrader almost in real-time.

5.3.2 Emulation of the Advanced Network Parameters and Effects

A major drawback of Dummynet is the lack of support for the advanced network parameters (delay jitter, packet reordering, packet duplication and packet corruption).

We therefore investigated and integrated KauNet into the NESSEE Server. Its basic pattern concept allows the user to manipulate network traffic either data-driven or time-driven. There are predefined pattern types for a precise reconfiguration of the data rate and the packet delay. Another pattern type is used to introduce a more precise mechanism for packet losses. Finally, it is possible to reorder packets by either the dedicated reordering pattern or by appropriate configurations of the delay change pattern.

Since there is no option within KauNet which allows duplicating packets, we added this feature on our own. We added a new duplication pattern type which behaves exactly like the already implemented ones. This means, one can decide between data-driven or time-driven mode. During our tests we noticed that it is not possible to remove patterns from the emulator’s configuration once they were added. This feature was added in our customized version as well.

The NESSEE Server was extended to fully support KauNet meaning the creation of pattern files and the proper configuration of pipes with these patterns. We decided to change the mechanism used to introduce packet loss. Dummynet’s approach is based on random numbers and is not highly precise. The pattern based approach increased the reproducibility of the experiments and the accuracy especially for shorter time periods.

One effect which cannot be configured by KauNet itself is the delay jitter. A special delay change pattern is created by the NESSEE Server to emulate this effect anyway. Therefore, a random number generator is used which can create values for different distribution functions. It generates delay times for every single packet position of a pattern file. Currently we support uniform, normal and Laplace distribution where the last one seems to be the most common form. [Zheng et al, 2001]

6. EVALUATION

In this section we investigate on the accuracy of the network emulation and show that the Degrader can handle the emulation of complex network behavior. The accuracy and performance of the underlying Dummynet itself was evaluated before and found to be reasonable [Nussbaum and Richard, 2009]. Therefore, we focus on the evaluation of the way our NESSEE Degrader uses Dummynet to emulate complex network topologies.

To validate that the Degrader really emulates the configured network parameters we determine the deviation of configured and actually measured values. Measuring delay is done with the ping[1] tool. Bandwidth and loss are determined with iperf[2]. The measured values are rechecked with the help of the statistics module of the Citrix Online video server, which also

---

[1] http://linux.die.net/man/8/ping

[2] http://sourceforge.net/projects/iperf/

collects information about bandwidth, loss and delay of the conference attendees. All tests are done with video conferences of ten minutes length.

To determine the emulation accuracy of the parameters independently from each other we defined various example scenarios in which either bandwidth, loss or delay is specified. In our measurements the maximum deviation of delay is +0.3ms (+0.2%) and the maximum deviation of loss is ±0.3%. To measure the bandwidth accuracy we had to create a test case without any bandwidth sharing between the nodes. We achieved an accuracy of ±3.5% of the configured bandwidth. We can observe that the amount of the configured values and the measured deviation correlate. This correlation has already been investigated by [Nussbaum and Richards, 2009].

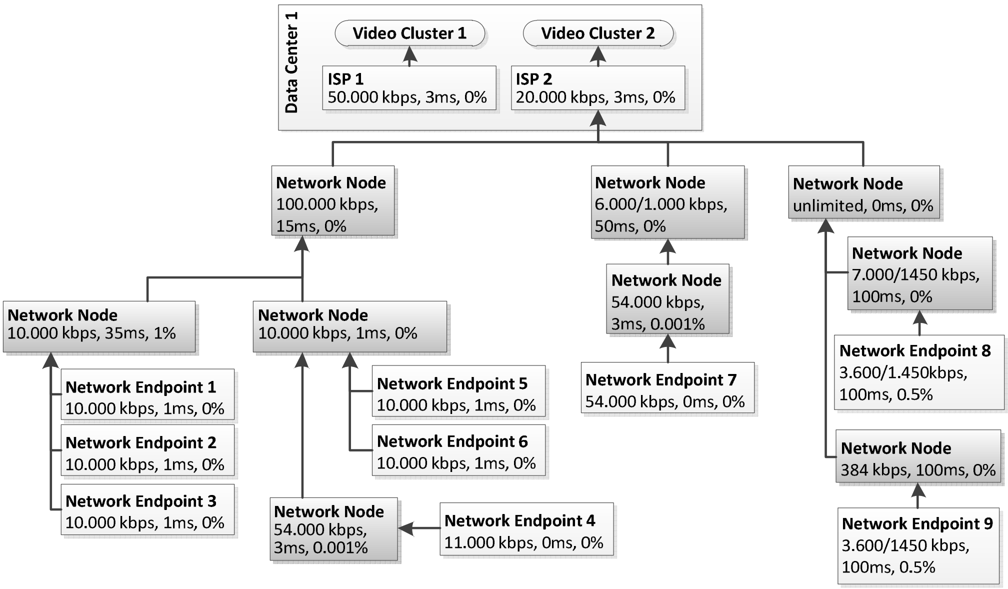

Figure 4. Network topology of the complex test. The configured network capabilities are bandwidth, delay and loss rate.

In real scenarios all parameters are configured at the same time. Therefore we analyzed the accuracy of all combined parameters in a more complex scenario. The topology of this scenario is illustrated in Figure 4. Nine Network Endpoints (NE), running the client SUT, can be found in this test. All of them communicate with one server cluster in a data center, which is connected to the Internet via an ISP with predefined network capabilities (bandwidth, delay, loss rate). The Network Endpoints are located in a network topology consisting of routers and gateways that are represented by Network Nodes. On the one hand, the expected result of this test is to see that bandwidth has to be shared, for example between Network Endpoints 1, 2 and 3. On the other hand we expect the different network capabilities to be combined. That means the delays of the single nodes should be added, the resulting bandwidth should be the minimum of the network path and the overall loss rate $OWLR_{sum}$ is determined according to Equation (1), where n is the length of the network path and $OWLR_i$ are the loss rates of the nodes.

$$OWLR_{sum} = 1 - \left(\prod_{i=1}^{n} (1 - OWLR_i) \right) \qquad (1)$$

Figure 5 shows the accuracy in the combined measurements. The loss rate is still emulated very accurately (±0.1%), but the deviation of delay has a maximum value of about 4ms. This is caused by the more complex network topology as the packets have to pass multiple pipes. Due to Dummynet's packet release mechanisms which may release packets slightly too early or slightly too late, the delay deviation may vary around the configured value in positive and negative direction.

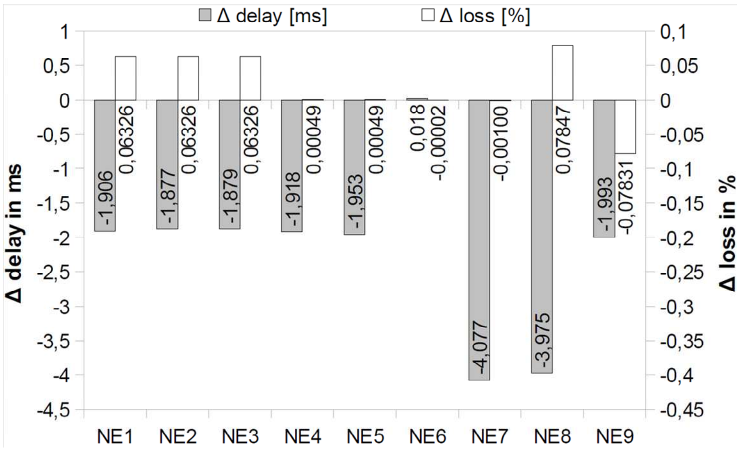

Figure 5. The deviation of configured and measured delay and loss values for each Network Endpoint (NE) of the topology shown in Figure 4

Regarding bandwidth sharing we can state that the measurements reflect the expected behavior. The bandwidth sum of NE1, NE2 and NE3 as well as NE4, NE5 and NE6 are nearly the same as the limitations of 10,000 kbps of the corresponding parent Network Nodes. The overall bandwidth of all Network Endpoints also meets the ISP2 bandwidth limitation of 20,000 kbps.

To evaluate our own KauNet modification that allows packet duplication we used the `ping` command to send and receive ICMP ECHO packets. In both possible modes, time- and data-driven, we configured a one way duplication rate of 40% and sent 100,000 packets. The results shown in Table 1 can be considered as very accurate. In data-driven mode exactly one duplicated packet seems to be missing. This is the duplicate of the last packet that is created correctly by the Degrader, but it is not recognized by the `ping` command as it already ends when the first response of the last request comes in. So the measured values exactly meet the expectations. In time-driven mode slightly more packets get duplicated. The reason for this is that the packets reach the Degrader in different time intervals. Therefore, it cannot be guaranteed that the packets hit the exact points in time in which the duplication is performed.

Table 1. Measurement results for the evaluation of packet duplication

	Time-driven mode		**Data-driven mode**	
	Expected	Measured	Expected	Measured
Sent requests	100,000	100,000	100,000	100,000
Received responses	140,000	140,189	140,000	139,999
Duplicate ratio	40%	40,189%	40%	39,999%

The duplication pattern was also evaluated inside a topology. When *n* cascaded nodes are configured with a certain duplication rate DR_i, the resulting duplication rate DR_{sum} can be determined using Equation (2).

$$DR_{sum} = \left(\prod_{i=1}^{n} (1 + DR_i) \right) - 1 \qquad (2)$$

The deviation of measured and expected DR_{sum} values for $n=3$ is ±0,25%. Similar values can be observed for the resulting loss rates $OWLR_{sum}$, if the pattern-based approach is used. For this reason the NESSEE Degrader emulates all pattern-supported parameters with KauNet and the remaining ones with Dummynet.

As we pointed out before the delay jitter is emulated according to a Laplace distribution. Table 2 shows the measured and expected delay values (minimum, maximum and mean) in a `ping` measurement with 20.000 packets and three cascaded network nodes. The mean measured values are slightly above the expected values, because of the delay that also exists without emulation. The expected minimum and maximum values are not always hit because of the minimal probability due to the Laplace distribution.

Table 2. Measurement results for the evaluation of delay jitter

		Upstream		**Downstream**	
		Expected	Measured	Expected	Measured
Delay variance ΔOWD [ms]	Min	28	32,094	28	31,849
	Mean	48	48,911	48	48,926
	Max	68	65,500	68	66,054

The reordering ratio is measured with iperf using UDP packets. If reordering RR_i is configured for *n* nodes inside a topology, packets can get reordered multiple times, which can result in a correct final order. Therefore the resulting reordering rate RR_{sum} cannot be determined exactly, but we can only calculate an upper limit. This is shown in Equation (3). The deviation of measured and expected reordering rate RR_{sum} is ±0.13%.

$$PRR_{sum} \leq 1 - \left(\prod_{i=1}^{n} (1 - PRR_i) \right) \qquad (3)$$

7. CONCLUSION AND FUTURE WORK

The main contribution of our work is an approach for the emulation of distributed systems with multi-level network topologies and fine-grained network parameters. To realize our approach we proposed an architecture for network emulation in large-scale tests, a generic Test Description Language and the NESSEE Degrader, that emulates network characteristics and behavior. The evaluation showed the implementation's good accuracy and the feasibility of emulating complex scenarios.

Our future work covers an extended comparison of existing network emulation solutions (software and hardware) not only based on features, but also regarding their performance and accuracy. We further want to investigate on methods to determine practice-oriented values for the emulation of typical scenarios and network access technologies. The presented architecture in Section 5 uses only one Degrader for network emulation. In our ongoing work we are designing and implementing architecture extensions for larger scenarios in which multiple Degraders will be necessary.

ACKNOWLEDGEMENT

This work results from the NESSEE 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

Carbone, M. and Rizzo, L., 2010. Dummynet Revisited. *ACM SIGCOMM Computer Communication Review*, Vol. 40, No. 2, pp. 12-20.

Carbone, M. and Rizzo, L., 2011. An emulation tool for PlanetLab. *Computer Communications*, Vol. 34, No. 16, pp. 1980-1990.

Carson, M. and Santay, D., 2003. NIST Net: a Linux-based network emulation tool. *ACM SIGCOMM Computer Communication Review*, Vol. 33, No. 3, pp. 111-126.

Garcia, J. et al, 2007. KauNet: Improving Reproducibility for Wireless and Mobile Research. *Proceedings of the 1st international workshop on System evaluation for mobile platforms*. New York, NY, USA, pp. 21-26.

Garcia, J. et al, 2008. KauNet: A Versatile and Flexible Emulation System. *Proceedings of the 5th Swedish National Computer Networking Workshop*. Karlskrona, Sweden.

Hemminger, S., 2005. Network Emulation with NetEm. *linux.conf.au*, Canberra, Australia.

Kalita, H. and Nambiar, M., 2011. Designing WANem : A Wide Area Network emulator tool. *Proceedings of the Third International Conference on Communication Systems and Networks (COMSNETS)*. Bangalore, India, pp. 1-4.

Nambiar, M. and Kalita, H., 2011. Design of a new algorithm for WAN disconnection emulation and implementing it in WANem. *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*. New York, NY, USA, pp. 376-381.

Nussbaum, L. and Richard, O., 2009. A Comparative Study of Network Link Emulators. *Proceedings of the 2009 Spring Simulation Multiconference*, San Diego, CA, USA, pp. 85:1-85:8.

OneLab, 2012. Developing testbeds for the Future Internet. Online at: http://www.onelab.eu/

Peterson, L. et al, 2006. Experiences building PlanetLab. *Proceedings of the 7th symposium on Operating systems design and implementation*, Berkeley, CA, USA, pp. 351-366.

Prasad, R. et al, 2003. Bandwidth estimation: metrics, measurement techniques, and tools. IEEE Network Magazine, vol.17, no.6, pp. 27-35.

Schwerdel, D. et al, 2010. German-Lab Experimental Facility. *Proceedings of the 3rd Future Internet Symposium*, Berlin, Germany.

Schwerdel, D. et al, 2011. ToMaTo - a network experimentation tool. *Proceedings of the 7th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom 2011)*, Shanghai, China, pp. 1-10.

Vahdat, A. et al, 2002. Scalability and Accuracy in a Large-Scale Network Emulator. *Proceedings of the 5th symposium on Operating systems design and implementation*, New York, NY, USA, pp. 271-284.

White, B. et al, 2002. An integrated experimental environment for distributed systems and networks. *Proceedings of the 5th symposium on Operating systems design and implementation*, New York, NY, USA, pp. 255-270.

Zheng, L. et al, 2001. Characteristics of network delay and delay jitter and its effect on voice over IP (VoIP). *Proceedings of ICC 2001. IEEE International Conference on Communications*, Helsinki, Finland, pp. 122–126.