Paul Müller*, Dennis Schwerdel and Justin Cappos

ToMaTo a Virtual Research Environment for Large Scale Distributed Systems Research

***Paul Müller:** E-Mail: pmueller@informatik.uni-kl.de
Dennis Schwerdel: E-Mail: schwerdel@informatik.uni-kl.de
Justin Cappos: E-Mail: jcappos@poly.edu

Abstract: Networks and distributed systems are an important field of research. To enable experimental research in this field we propose a new tool ToMaTo (Topology Management Tool) which was developed to support research projects within the BMBF funded project G-Lab. It should support researchers from various branches of science who investigate distributed systems by providing a virtual environment for their research.

Using various virtualization technologies, ToMaTo is able to provide realistic components that can run real-world software as well as lightweight components that can be used to analyze algorithms at large scale.

This paper describes how an additional virtualization technology from the Seattle testbed has been added to ToMaTo to allow even larger experiments with distributed algorithms. Moreover the paper describes some concrete experiments that are carried out with ToMaTo.

1 Introduction

Many research projects focus on improving various aspects of distributed networks e.g. networking concepts and architecture of networks in general [MR08]. All of these research projects need ways to evaluate their ideas and results. Simulations can model reality, but will never show unforeseen behavior as real systems do. Therefore research environments aim to provide a realistic environment for experiments.

Realism is an important aspect of a virtualized environment it describes how similar the environment is to the real world. Some effects of reality are only visible in environments with a certain degree of realism, which restricts the kinds of experiments that can use the research environment. On the other hand it is important to use the resources efficiently and run as many experiments as possible on as few resources as possible since research environments are expensive in both acquisition and operation. There exists an obvious trade-off between parallelism and realism in practice. Real systems offer the best realism possible but only allow one experiment to use the resources at a time. Research environments can use virtualization technologies to achieve parallelism at the cost of losing realism.

Most existing research environments for distributed systems (testbeds) operate on a fixed level of realism and parallelism. A lack of parallelism results in highly limited resources so that only a small number of instances can be used across all researchers. Conversely, if an environment has a limited amount of realism, certain types of experiments cannot be run at all. Federations allow users to create experiments spanning testbeds with different degrees of realism and parallelism. This reduces resource wastage but also causes new problems since some testbed features are not usable across testbed boundaries.

The topology management tool (ToMaTo) [SHG+11] has been developed as part of the German-Lab project [Ger, SGH+10] to provide a virtualized research environment for networking experiments. ToMaTo uses a unique approach to reduce resource wastage: Users can select between multiple virtualization technologies for each component of their experiment. This way each component uses only the resources it needs to provide the realism that the experiment needs and thus parallelism is maximized and resource wastage minimized. ToMaTo effectively operates on multiple levels on the realism/parallelism scale as shown in Figure 1.

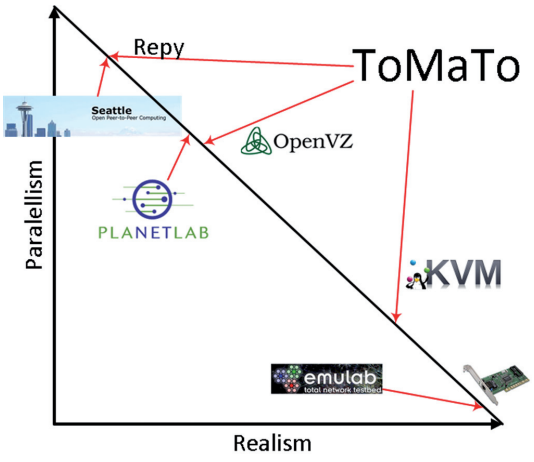

Figure 1: Realism/Parallelism scale.

This paper describes how the restricted python language (Repy) [CDR+10] developed as part of the Seattle testbed [CBKA09] has been added to ToMaTo as a new virtualization technology that offers extremely lightweight programmable devices. Section 2 analyses related work and discuss virtualization technology used by different testbeds. The ToMaTo testbed is described in Section 3 and Section 4 explains the integration of the Repy technology. Section 5 explains the usage of ToMaTo in experimentation and section 6 explains three experiments that have been conducted using ToMaTo before section 7 concludes.

2 Related work

Emulab [GRL05, ESL07] is a testbed that does not use virtualization technology for its hosts, so physical computers can be used for experiments. While this setup offers the best possible realism, it may waste resources since only one experiment component can be run on a host at a time. To mitigate this, Emulab also provides a small number of nodes that have operating system virtualization and may act as multiple different nodes. This turns one physical node into several *virtual nodes*. These virtual nodes have an increased parallelism since multiple instances can be run on one host, but the hosts are still exclusively used by one experiment. The virtual nodes also have reduced realism when compared to physical nodes.

The Planet-Lab testbed [PBFM06, PR06] uses container-based virtualization technology and offers its users access to Linux hosts with limited kernel-space access. This virtualization technology is a good compromise of realism and parallelism from an operating system point of view. Still, some experiments can not be executed if they need more access to the operating system or, more likely, the network configuration. The parallelism is much greater than dedicated environments like Emulab offers. However, resources are still wasted since the container-based virtualization runs one complete user space system for each virtual device.

The Seattle testbed [CBKA09] allows its users to execute Repy programs on the testbed nodes. Testbed nodes are actually multi-use devices like laptops, phones, desktops, etc. that are used by end users around the world. Repy programs can use limited memory and processor resources on the host computer, and network resources are shared by assigning different port ranges to different Repy programs. The Repy language and the restrictions enforced on Repy programs hugely limit the realism of the Seattle testbed. On the other hand, the Seattle testbed offers a very high parallelism. There are only few dedicated Seattle servers as the parallelism would be limited by port number restrictions, in fact most Seattle resources are obtained by volunteer computing.

Simulation tools like ns-3 [ns311], omnet [VH08] and mininet [LHM10] model reality to a certain degree and are able to run huge experiments. Simulations can modify the progress of time in the simulated environment and thus use the full resources of the host during the whole experiment run. Therefore simulators offer the best possible resource usage but also a very limited realism since the simulation environment contains only a model of the reality.

3 ToMaTo

ToMaTo is a topology-oriented networking testbed designed for high resource efficiency, i.e. high parallelism where possible but high realism where needed. Topologies consist of devices (produce and consume networking data) and connectors (manipulate and forward data). Devices contain a set of networking interfaces that can be connected to connectors. Figure 2 shows a simple topology consisting of five devices (one central server and four clients) and three connectors (two switches and one Internet connector). To increase both flexibility and resource efficiency,

ToMaTo offers different types of devices and connectors. Users can choose between hardware virtualization, which provides an environment nearly identical to a real computer but has a high resource usage, and container virtualization that uses fewer resources but does not suit all needs. The topology in Figure 2 uses hardware virtualization for the server since it runs a software that requires hardware virtualization. To save resources it uses container virtualization for the clients. This example shows that the testbed offers full flexibility on the one hand while also saving resources on the other hand.

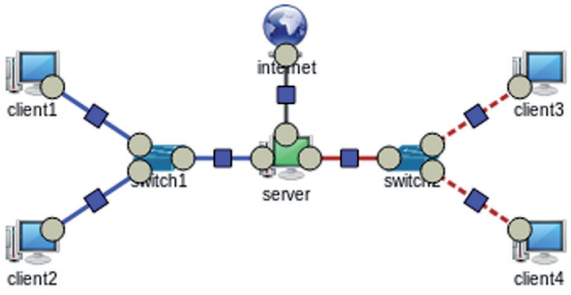

Figure 2: Example topology.

The default connector type is a VPN connector with a selectable forwarding policy (hub, switch or router). Public

services, cloud resources or even other testbeds can be combined with ToMaTo topologies by using external network connectors. To help users with running their networking experiments, ToMaTo offers an easy to use, web-based front-end with an intuitive editor. Users can control their devices directly from their browser or using a VNC client of their choice. Advanced tools like link emulation and packet capturing are included in ToMaTo and can be used to run and control experiments.

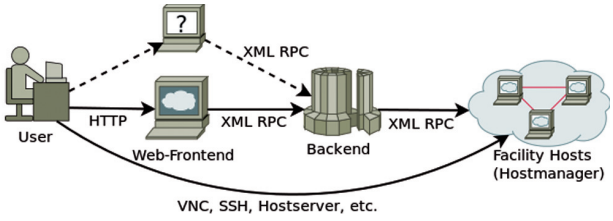

Figure 3: ToMaTo structure.

The ToMaTo software consists of three tiers as shown in Figure 3. The hosts provide virtualization technology and a complete toolset needed for advanced features like link emulation, packet capturing, etc. The back-end component contains all the control logic of the ToMaTo software and remotely controls the hosts. Different front-ends use the XML-RPC interface provided by the backend component. The most important front-end is the web-based user front-end that allows users to edit and manage their topologies from their browser using modern web technologies. Other front-end software includes a command-line client that allows easy scripting and an adapter for the Teagle federation framework [CMW10, WMFP11].

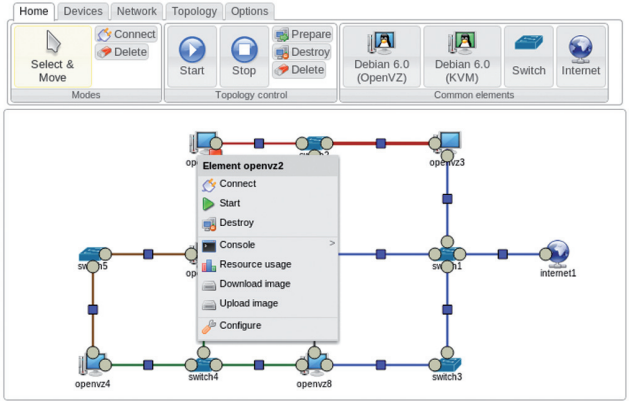

Figure 4: ToMaTo editor.

One of the key features of ToMaTo is its graphical user interface that allows even unexperienced users to create complex network topologies by drag and drop. ToMaTo features an easy-to-use editor (Figure 4) to create, manage, and control networking topologies. With this editor, users can configure their topology components and control them in the same intuitive interface. The editor also gives users direct access to their virtual machines using web-based VNC technology.

4 Repy

The Repy sandbox was created to isolate code for the Seattle testbed [CBKA09]. The major goals of the sandbox are to provide performance isolation, security isolation [CDR+10], and a portable programming interface across diverse device types. At a high level, a Repy sandbox is intended to be secure enough that one would allow a non experienced user to execute code within it. In this way, it is somewhat similar to existing sandboxes like Flash and Java, except these are meant to execute code from visited webpages and the like. The Seattle model intends that resources will be used in more of a cloud like manner, so that unknown parties may push code to a user's machine on demand. However, the attack surface is minimized, in part by restricting the sandbox functionality.

Clearly, performance isolation is also an important goal for Repy. A Repy sandbox is meant to be lightweight yet have the same sort of performance isolation that exists in commercial operating system VM offerings. To achieve this, Repy uses operating system hooks commonly used for debugging to vary the amount of CPU and memory available to a sandboxed program. To restrict other resources such as network and disk I/O, Repy checks calls that access these resources and prevents or delays their execution if they go over a configurable quota. This allows the performance impact of a Repy program to be selected and controlled. (For our purpose, this allows the amount of parallelism to be very high while retaining realistic performance isolation amongst instances.)

The Repy sandbox is based upon Python and consumes few resources. Running a hello-world Repy program takes about 1 MB of memory on most computers. The performance impact seen by running code in Repy depends on the operation, but averages about 7% over native Python on most benchmarks.

While Repy's interface was designed explicitly to minimize and secure the attack surface, it was also designed for extensibility in two ways. First, a mechanism similar to systemcall interposition (or object capability systems) is built into Repy. This allows one to enforce policies upon a Repy sandbox without needing to change the trusted computing base [CDR+10]. For example, if one wants to restrict

access to certain network destinations or sources, this can be added in a secure manner without changing Repy.

Repy itself was also designed to be extendable by adding new functionalities. Adding access to a new type of device or changing the level of access to a device requires a few minor modifications to the code. One simply codes the new routines, connects them into a module that performs a security validation on data passing through it, and this is now available to Repy programs running in this modified sandbox. This was leveraged to integrate Repy with ToMaTo.

4.1 Integration

ToMaTo's structure and its support for different device and connector types make it very flexible and extensible. The basic interaction point that all devices and connectors must support are network bridges where virtual network interfaces can be connected. New element types can be easily added to ToMaTo as long as they fit into the basic topology model (devices with interfaces, connectors and connections) and support bridges as interaction points. All other features are optional and can be described by capability records. Thus, the integration of programmable devices as a new device type was easily possible with minimal changes to ToMaTo.

The first part of the integration of Repy as programmable device into ToMaTo was adapting Repy and thus creating ToMaTo-Repy. In the Seattle testbed, the interface for Repy scripts contained methods to access the network via UDP/IP and TCP/IP, to read and write files in a dedicated folder and some utility methods to retrieve the time, set timers etc. These methods needed to be exchanged with network interfaces speaking Ethernet to match the ToMaTo model.

ToMaTo-Repy extends Repy and adds the capability to create and connect to virtual network interfaces (VNIs). The names of the VNIs and their alias seen by Repy-scripts can be set via command-line arguments. The programming interface exposed to the repy-scripts has been extended to include methods to list available VNIs, send data on a specific VNI and receive data from one or all VNIs. The method to receive data allows non-blocking calls as well as blocking calls with a configurable maximum timeout. To avoid polling, a special method is exposed that listens on all VNIs at the same time and returns the received data as well as the name of the VNI it has been received on.

The data units that scripts in ToMaTo-Repy work with are raw Ethernet packets. The scripts are free to parse the headers according to Internet standards or to use custom protocols to speak to other devices. Also the networking behavior (i.e. protocol handling) can be completely defined by the scripts, and there is no kernel that will process the Ethernet packets. Despite the name "programmable device", users can also use these devices to implement connectors and middleboxes like firewalls, routers, etc.

To simplify the usage of ToMaTo-Repy when working with Internet protocols, the struct library has been added to the interface for scripts to help encoding and decoding binary data structures. Also a collection of ToMaTo-Repy scripts containing implementations of several Internet protocols is publicly available and constantly extended to help users program their programmable devices.

ToMaTo-Repy has been packaged for Debian-based systems like the ToMaTo hosts and is installed during setup of the hosts. The backend component of ToMaTo has been extended to include ToMaTo-Repy as a third device type called programmable device. Users can configure parameters that are passed to the script upon execution. This way the same script can be used for multiple devices and variable data like addresses can be configured separately. The virtual machine based devices allow users to upload and download the device image, i.e. the virtual harddisk. For programmable devices this paradigm is used to upload and download the ToMaTo-Repy scripts to/from the devices. Also a collection of ready-to-use images, called templates, is available for programmable devices like it is for the other device types. These templates include a pingable node, a switch, a DNS server, and a DHCP server. The output of the device's script is directed to a log file and can be viewed using the same VNC technology used to access the consoles of virtual machine based devices.

4.2 Evaluation

The introduction of programmable devices into ToMaTo using Repy from the Seattle testbed allows users to use networking devices in their experiments that can be easily programmed but do not have the resource usage and overhead of virtual machines.

ToMaTo-Repy offers users to program protocols at Ethernet level with the simplicity of the high-level programing language python. The library "tomatolib" contains a collection of protocol implementations and code snippets written in ToMaTo-Repy that can be used to write complex programs. Table 1 shows the code sizes of example programs by counting non-empty lines of code of the script Lines of code Script Library Total defining the program, the included parts of tomatolib and the total lines of the combined script.

Lines of code	Script	Library	Total
Pingable node	12	242	254
Switch	5	29	34
DNS server	22	377	399
DHCP server	21	386	407
partial IRC server	27	382	409

Table 1: Source code size examples.

Besides simplicity, resource usage and performance are important aspects of the new device type. The resource usage of the programmable devices depends on the script that is being executed. For most scripts, the overhead of the python interpreter and the Repy sandbox is much more than the resources needed by the script itself. The amount of RAM that is being used has been measured to be between 5 and 10 megabytes, but of course this value depends on the software versions and the script. The Repy sandbox has the ability to restrict the resource usage of the scripts it executes, which could be used to limit the resource usage when needed.

Since Repy runs in user space, additional context switches are needed for packet processing. Also python is expected to perform worse than the highly optimized kernel code written in C. A ping experiment has been run to determine the additional overhead of a Repy node compared to pinging interfaces of the virtual machine based devices. Two different Repy nodes have been implemented that both are able to respond to ARP requests and to ICMP echo requests. The first implementation hierarchically dissects the packet headers and processes them in a modular way while the second implementation picks only those byte ranges from specific offsets that are needed to detect the request and recombines them to form the answer. Figure 5 shows the CDF and the median RTT of the different implementations and device types. Repy performs better than the KVM hardware virtualization since the Repy script does not have the overhead of a complete kernel. The OpenVZ container virtualization is able to beat all other device types because it handles the pings inside a different namespace of the host kernel without any context switches. Figure 5 shows, that the Repy programmable device introduces an overhead between 0.1 and 0.15 milliseconds due to context switching and the python language. Notably the optimized Repy implementation was able to reduce the overhead significantly. Even for the normal implementation, the overhead introduced is low enough to allow networking experiments with precise link emulation. Since version 2.1, the programmable device is an integral part of ToMaTo and users can use ToMaToRepy scripts to build complex yet resource efficient devices. Experiments have shown that the number of programmable devices that can run concurrently on one host is at least 1000 but these values depend on the activity and complexity of the scripts. ToMaTo-Repy helps to extend the realism/parallelism range of ToMaTo and thus improve its flexibility and efficiency.

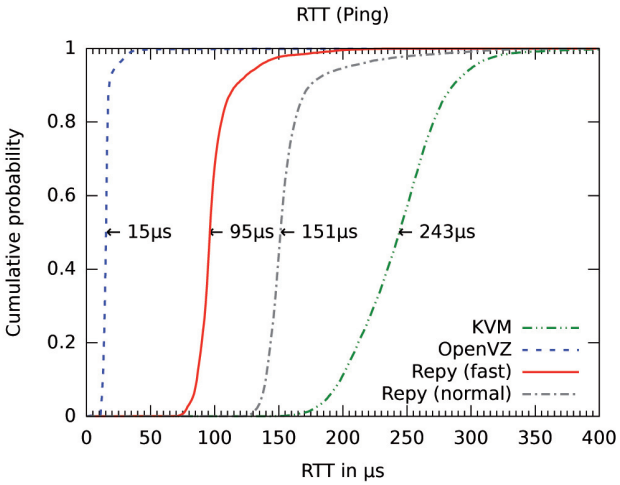

Figure 5: CDF of RTT experiment.

5 Experimentation

Evaluating distributed algorithms is a complex task. One approach is to compare the design goals or requirements with the actual capabilities of the resulting software. In case of an experimental facility the design goal is to support experiments and help researchers carry out their experiments. To evaluate ToMaTo based on this goal section 5.1 first develops a classification of experiments in the German-Lab project and section 5.2 outlines how ToMaTo supports these types of experiments. Section 5.3 takes a quick look at the efficiency and scalability of ToMaTo.

5.1 Types of experiments

In general four different experiment types can be identified in networked respectively distributed systems research.

Access layer experiments. Access layer experiments consider the lower networking layers and examine the usage of hardware for networking. An example for this experiment class is mobile handover protocols. These experiments need access to real hardware, they often need to run custom operating systems (e.g. with real-time support) and they need heterogeneous access technologies (3G, IEEE 802.11, Fiber, etc.). In most cases, these requirements can

only be fulfilled with custom testbeds, thus supporting this kind of experiment was not a design goal for ToMaTo.

Network layer experiments. These types of experiments consider the TCP/IP suite and its networking layers. Examples for this class are experiments with IPv6 extensions and TCP substitutes. This kind of experiment needs to run modified kernels. The resources that a single experiment needs are normally limited to a few devices but these devices have to be connected in complex network topologies with link emulation.

Protocol/Algorithm experiments. Experiments with protocols or algorithms work on top of the network layer and usually consider new approaches for lager networks. Nearly all peer-to-peer experiments fall in this category. These experiments need a high number of nodes but usually no hardware or kernel access. They only need simple network topologies with link emulation.

Legacy application experiments. Experiments using legacy software cannot be modeled because of its unspecified or unpublished behavior. Examples of this software are Skype and Windows. The experiments with this software often need special operating system environments including Internet access and link emulation. In turn, these experiments normally do not need big or complex network topologies.

Experiences of the German-Lab experimental facility show that most experiments can be categorized fairly well with this scheme. A few experiments have two experiment classes, and thus have requirements of both classes. The resource requirements of the classes are very heterogeneous but a general tradeoff between more resource access and access to more resources becomes evident, i.e. in general experiments either need a high number of (lightweight) resources with restricted access or a small number of resources with full access.

5.2 Experiment support in ToMaTo

ToMaTo has been designed to support all experiment classes identified in section 5.1 except for access layer experiments because these experiments need a specialized testbed depending on the access technology. The Wisebed [BCD+10] and DES testbed [Des10] for example are specialized experimental facilities for sensor networks and wireless networks.

Network layer experiments can be done easily in ToMaTo using KVM devices and virtual switches. The KVM devices over all need flexibility in kernel choice and modification required by this experiment class. Switched networks provide connectivity on layer 2 so that any TCP/IP modification or substitute can be examined. Using the command line frontend even very complex topologies can be easily designed. The possibility to capture and download network traffic can be very handy for this kind of experiment.

Protocol/Algorithm experiments are supported in ToMaTo using OpenVZ devices. Since OpenVZ devices are very lightweight, a high number of devices can be used in topologies. Using an Internet connector, external resources like PlanetLab nodes can be included in the experiment. Using the upload/download image feature, users can prepare a device image once and upload it to all of their devices. Capturing network traffic and link emulation can be used to debug the protocols.

ToMaTo also supports legacy application experiments using KVM devices and Internet connectors. KVM devices can run nearly all x86 operating systems including Windows and BSD, therefore users can build custom environments for their legacy applications. The legacy application can communicate with external services using the Internet connector. Traffic of the legacy application can be captured and analyzed using specialized tools without any operating system support.

5.3 Efficiency and scalability

With ToMaTo, users can choose between OpenVZ and KVM virtual machines. This way, users can get the level of access that is needed for their experiments and still use as few resources as possible. A single cluster node can handle up to 250 OpenVZ devices and up to 50 KVM devices, both depending on device usage. The networking components only pose a very small overhead and can handle connections with over 100 Mb/s without influencing them.

ToMaTo hosts use an existing operating system as basis and only need small changes that have been bundled as a software package. This has the advantage that operating system support and security updates are available from the original sources and do not have to be provided by the experimental facility administrators. As the ToMaTo back-end only controls the hosts and only contacts them when users change their topologies, the back-end can handle many host nodes making the solution very scalable.

ToMaTo can be used to create experimental facilities with distributed hosts. Limitations in network emulation apply since the resulting link characteristics are a combination of real and emulated link properties. ToMaTo offers

long-term link statistics so the users can plan their experiments accordingly. ToMaTo allows its users to design, manage and control networking topologies for use in network research. ToMaTo fits for a wide range of experiments identified as common in the context of the German-Lab project. The design of the testbed software offers efficiency and scalability. ToMaTo is not bound to German-Lab and can easily be used to build similar experimental facilities.

In the German-Lab testbed currently 35 of 182 hosts are ToMaTo-enabled. The goal is to increase this number also by integrating nodes from international projects and thereby increase the usability of the testbed. Also the integration of OpenFlow is already implemented.

6 Experiments in ToMaTo

6.1 Experiment: German-Lab Deep

ToMaTo has been successfully used for an experiment in the context of German-Lab Deep [KVS+11] project that focuses on attack mitigation in VoIP systems. In this scenario, an attacker initiates malicious calls that must be detected by the distributed VoIP system and blocked at the nearest gateway using cross-layer components. Figure 6 shows the topology used for this experiment. It contains several end systems and smart middle boxes using the KVM technology. In this scenario, it was very important to explicitly define the links in the topology and to be able to route traffic over programmable middle boxes. This experiment was central to the whole project and brought together different software components. A demonstration at the Euroview conference successfully showed the interoperability of these components using ToMaTo [KVS+11].

6.2 iGreen Mobile Application Test

The research project iGreen aims to introduce location-based semantic services in the agricultural industry in order to make use of data available in the public or private sectors directly in the field. Using Internet-based services in rural areas is a challenging task because of the lack of highquality network coverage for WiFi or even cell phones. The use of satellite connections with high delay as well as the low availability of network coverage for cell phones imposes interesting requirements for testing mobile applications used in the agricultural sector.

The iGreen mobile application (shown in Figure 7) is designed to offer access to various iGreen services supporting the decision making process related to agricultural tasks like proper fertilization or application of pesticides. This application will run on mobile devices in the field or as embedded software on farm machines. In both cases, the application has to access services that run in a compute cloud using whatever network connection is available at its location, i.e., connectivity by a mobile carrier or by a direct satellite link.

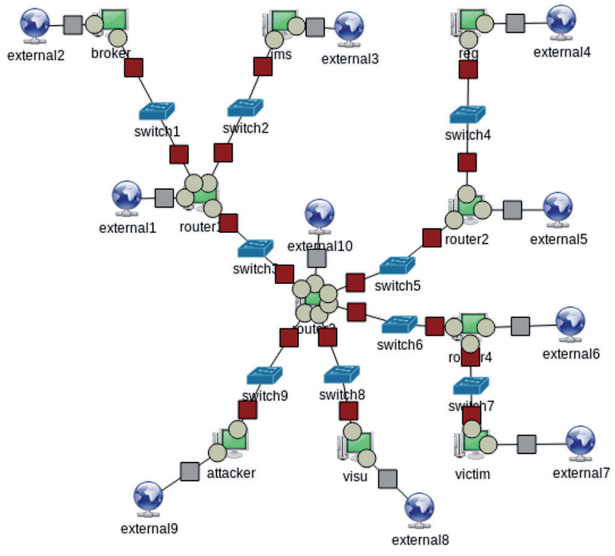

Figure 6: The topology used by a DeepG experiment. It contains an attacker, a victim and several components needed to mitigate this attack. Smart routers connect all components.

Figure 7: The iGreen mobile application.

In an experiment using ToMaTo [SGR12], the prototype version of the mobile iGreen application has been tested in different network connectivity scenarios to analyze the influence of delay and packet loss on the service quality. In this scenario, the mobile application runs in an emulator, embedded in a ToMaTo topology that contains a configurable link and a bridge to the Internet where the iGreen service resides. The test has been carried out with different link properties, matching the characteristics of UMTS, GPRS and Satellite links. Hence different implementation problems resulting in a degraded service quality have been identified. For example, the software retrieved the messages by first requesting a list of message IDs and then requesting each message synchronously in a loop. This resulted in huge delays due to the accumulated round-trip-times that could be reduced by requesting all messages asynchronously in parallel.

6.3 Malware Analysis Experiment

A live demonstration at the Euroview conference 2011 [SRM11] showed the unique features of ToMaTo that allow users to analyze network behavior of malicious software. In this demonstration, the network behavior of an older Internet worm has been analyzed using ToMaTo. Figure 8 shows the topology that has been used in the demonstration. It clearly shows that the Internet in the upper left corner is not connected to the rest of the topology, so all the other components of the topology had no access to the Internet and the worm could safely be analyzed without the risk of escape. In most other networking testbeds, this would not be possible, because there is an implicit network link to the Internet or to core components of the testbed.

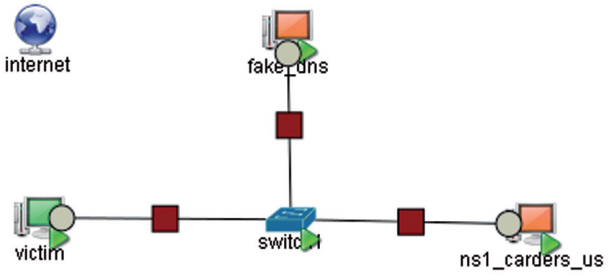

Figure 8: The topology used for a malware analysis experiment. This topology is not connected to the Internet for safety reason.

In the analysis of the malware, first a virtual machine using KVM and running Windows XP has been configured. A backup of that machine has been downloaded to be able to replay the infection later. Using the packet-capturing feature on the outgoing link and scripted network components it was possible to analyze the network behavior of that software without relying on tools that run inside the infected machine.

As a result of the analysis as shown in Figure 9, the control server has been identified as a probably hacked name server. The protocol used by the control server could be identified as IRC and even the channel could be captured. With this information it would be possible to contact the povider of the control server and shut it down or to try a man-in-the-middle attack on the control protocol for further analysis.

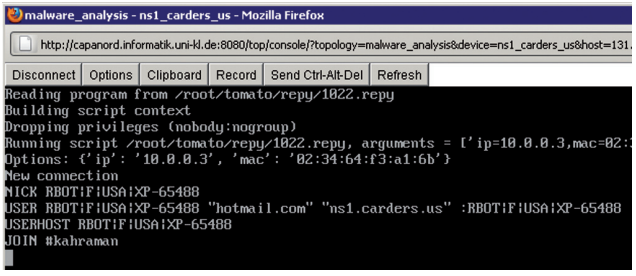

Figure 9: The results of the malware analysis reveal that a hacked name server is used as a master server that controls the victims using IRC.

7 Conclusion

To conclude, this paper describes how the Repy sandbox from the Seattle project was integrated with ToMaTo. This allows ToMaTo to run experiments that have a high degree of parallelism, while reducing the realism. By combining Repy together with the other virtualization technology, ToMaTo can obtain a much wider range of parallelism/realism.

This result shows how well-suited ToMaTo is for distributed systems research since it can provide realistic components that can run real-world software as well as lightweight components that can be used to analyze algorithms at large scale. This flexibility allows ToMaTo to use much less resources than other comparable testbeds in most scenarios.

Since the research projects based on German-Lab ended, the GLab facility including the ToMaTo testbed is open for external users. Interested researchers can contact the authors or visit the webpage at http://www.tomato-lab.org for information on accessing the testbed.

The integration of the Repy technology shows how easy it is to integrate new technology into ToMaTo be-

cause of its flexible architecture. ToMaTo developers are always open for external contributions and joint research projects aiming towards further ToMaTo development and usage.

References

[BCM+10] Tobias Baumgartner, Ioannis Chatzigiannakis, Maick Danckwardt, Christos Koninis, Alexander Kröller, Georgios Mylonas, Dennis Pfisterer, and Barry Porter. Virtualising testbeds to support large-scale reconfigurable experimental facilities. In Proceedings of EWSN – 7th European Conference of Wireless Sensor Networks, pages 210–223, 2010.

[CBKA09] Justin Cappos, Ivan Beschastnikh, Arvind Krishnamurthy, and Tom Anderson. Seattle: a platform for educational cloud computing. In *Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2009*, pages 111–115, 2009.

[CDR+10] Justin Cappos, Armon Dadgar, Jeff Rasley, Justin Samuel, Ivan Beschastnikh, Cosmin Barsan, Arvind Krishnamurthy, and Thomas E. Anderson. Retaining sandbox containment despite bugs in privileged memory-safe code. In *ACM Conference on Computer and Communications Security*, pages 212–223, 2010.

[CMW10] K. Campowsky, T. Magedanz, and S. Wahle. Resource management in large scale experimental facilities. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 930–933, april 2010.

[Des10] DES-Testbed A Wireless Multi-Hop Network Testbed for future mobile networks, Stuttgart, Germany, 06/2010 2010.

[ESL07] Eric Eide, Leigh Stoller, and Jay Lepreau. An experimentation workbench for replayable networking research. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, NSDI'07, pages 16–16, Berkeley, CA, USA, 2007. USENIX Association.

[Ger] German-Lab Project. German-Lab Website <http://www.german-lab.de>.

[GRL05] Shashi Guruprasad, Robert Ricci, and Jay Lepreau. Integrated Network Experimentation using Simulation and Emulation. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, pages 204–212, Washington, DC, USA, 2005. IEEE Computer Society.

[KVS+11] Michael Kleis, Christian Varas, Abbas Siddiqui, Paul Müller, Irfan Simsek, Martin Becke, Dirk Hoffstadt, Alexander Marold, Erwin Rathgeb, Christian Henke, Julius Müller, and Thomas Magedanz. Cross-layer security and functional composition for a future internet. Proceedings of 11th Würzburg Workshop on IP: Joint ITG and Euro-NF Workshop "Visions of Future Generation Networks" (EuroView2011), 2011.

[LHM10] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets '10, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[ns311] NS3 The ns-3 network simulator. http://www.nsnam.org/, Accessed 2011.

[PBFM06] Larry L. Peterson, Andy C. Bavier, Marc E. Fiuczynski, and Steve Muir. Experiences Building PlanetLab. In *OSDI*, pages 351–366, 2006.

[PR06] Larry L. Peterson and Timothy Roscoe. The design principles of PlanetLab. *Operating Systems Review*, 40(1): 11–16, 2006.

[MR08] Bernd Reuther and Paul Mueller. Future Internet Architecture – A Service Oriented Approach. *it – Information Technology*, 50(6), 2008.

[SGH+10] Dennis Schwerdel, Daniel Günther, Robert Henjes, Bernd Reuther, and Paul Müller. German-Lab Experimental Facility. In *Proceedings of FIS 2010 – Third Future Internet Symposium*, pages 1–10, 2010.

[SGR+12] Dennis Schwerdel, Joachim Götze, Bernd Reuther, and Paul Müller. Testing mobile apps in the tomato testbed. Proceedings of 12th Wuerzburg Workshop on IP (EuroView), 2012.

[SHG+11] Dennis Schwerdel, David Hock, Daniel Günther, Bernd Reuther, Paul Müller, and Phuoc *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, April 2011.

[SRM11] Dennis Schwerdel, Bernd Reuther, and Paul Müller. Malware analysis in the tomato testbed. Proceedings of 11th Wuerzburg Workshop on I (EuroView), 2011.

[VH08] András Varga and Rudolf Hornig. An overview of the OMNeT++ simulation environment. In *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[WMFP11] S. Wahle, T. Magedanz, S. Fox, and E. Power. Heterogeous resource description and management in generic resource federation frameworks. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 1196–1199, may 2011.

Paul Müller: University Kaiserslautern,
Computer Science Department,
Paul Ehrlich Street Bld. 34,
67663 Kaiserslautern/Germany

Dennis Schwerdel: University Kaiserslautern, Computer Science Department, Paul Ehrlich Street Bld. 34, 67663 Kaiserslautern/Germany

Justin Cappos: Polytechnic Institute of New York University, Six MetroTech Center, Brooklyn, NY 11201, USA