ToMaTo - a network experimentation tool

Dennis Schwerdel[1], David Hock[2], Daniel Günther[1], Bernd Reuther[1], Paul Müller[1] and Phuoc Tran-Gia[2]

[1] Integrated Communication Systems Lab, University of Kaiserslautern, Germany
`{schwerdel,guenther,reuther,pmueller}@informatik.uni-kl.de`
[2] University of Würzburg, Germany
`{david.hock,trangia}@informatik.uni-wuerzburg.de`

Abstract. Networks are an important field of research in information technology and experimental facilities are key instruments to enable practical research in this area. In the context of the German-Lab project, the topology management tool "ToMaTo" has been developed to be used as experimental facility software. This paper describes the features and the design of ToMaTo and evaluates it by identifying common experiment types and matching their requirements to the properties of ToMaTo.

1 Introduction

The Internet has a large economic influence but is based on legacy mechanisms and algorithms from the 70's and 80's. The rapid evolution of applications and transport technologies demands for changes even of core technologies of the Internet. A lot of research work has been done on improving isolated aspects of the Internet but in the last years also a lot of holistic research efforts investigate concepts and technologies for future networks in general[7].

All of these research projects need ways to evaluate their ideas and results. In the beginning of the projects, theoretical models and simulations might be sufficient but at some stage a more realistic environment is needed. Real networks and real hardware show unforeseen effects that cannot be modeled. New protocols and architectures will have to work with legacy components, i.e. currently widespread hardware and software, which have often unpublished behavior details.

Experimental facilities aim to provide a realistic environment for experiments using emulation techniques. In experimental facilities, there is always a trade-off between realism, concurrency and repeatability. Realistic environments show unforeseen and random effects that cannot be repeated. To be able to run concurrent experiments on the facility, the access of each experiment must be restricted to sharable or virtualized resources which in turn limits the realism.

A lot of software for experimental facilities has been developed and each one works at a certain level of realism, concurrency and repeatability. The German-Lab experimental facility allows its researchers to choose from various experimental facility software. An experimental facility software called Topology Management Tool (ToMaTo) has been developed in the German-Lab project. ToMaTo allows researchers to create virtual network topologies populated by virtual nodes running standard software.

This paper describes the ToMaTo software and compares it to other experimental facility software. Section 2 gives an overview of other experimental facility software and comparable solutions. The design of ToMaTo is described in section 3. Section 4 evaluates the design by identifying common experiment types and outlining the support for these experiments in ToMaTo. Section 5 concludes the work and mentions future work on ToMaTo.

2 Related work

Network experimentation tools like VIRCONEL[4] and the Common Open Research Emulator (CORE)[2] can setup and control virtual machines connected by a virtual network topology. Both tools use virtualization to run multiple virtual computers on a physical host and they use tunnels to create a virtual network topology. CORE also allows to configure emulated link characteristics. CORE and VIRCONEL were created to allow a single user to setup an experiment, so they lack the ability to handle multiple users or multiple concurrent experiment topologies. They have additional limitations when used across multiple hosts.

On the other hand there are network research testbeds that allow multiple researchers to run experiments concurrently. The most well-known of them is probably Planet-Lab[8], which uses a container-based virtualization to run multiple virtual computers on a single host. Planet-Lab has a very large number of distributed hosts and thus is well-fitting for peer-to-peer experiments. Due to its container-based approach, Planet-Lab cannot support kernel-space modifications or other operating systems. Originally Planet-Lab does not offer any way to its users to configure the network topology. An extension called Vini tries to improve in this area but the container-based virtualization technology and high distribution of the hosts pose limits on that.

Another well-known testbed is Emulab[10]. Emulab is a highly heterogeneous testbed with wifi and radio components as well as distributed nodes but the core is a computing cluster that allows users to boot custom software on the cluster nodes and connect them with virtual network topologies provided by dedicated hosts. This setup allows researchers to access actual hardware and the virtual networks offer high bandwidth. On the other hand the testbed design does not support distribution of physical hosts and efficiency is low because all experiments use real hosts as no virtualization is used.

Seattle[5] is a novel peer-to-peer testbed mainly targeted towards network algorithm testing. The testbed software consists of a custom python interpreter and management modules, spread across the world by volunteer computing. Users can run algorithms written in a custom python dialect on virtual machines across the testbed. Since Seattle only supports software that has been written in its custom python dialect, it is not capable to run any existing software.

Wisebed[3] and the DES testbed[1] are specialized experimental facilities for sensor networks and wireless networks. Their design is mostly defined by the special needs of the hardware. Virtualization and distribution is limited by the capabilities of the physical hosts.

ToMaTo's goal is to overcome limitations found in experimental facility software so that the user has maximal flexibility for his experiments. ToMaTo allows its users to configure and use multiple concurrent network topologies. It also aims to allow lightweight virtualization and full operating system access for the experiments.

3 ToMaTo design

The goal of ToMaTo is to enable users to create and use network topologies for their experiments. A network topology consists of two types of components. Devices are active components like computers that run the software of the experiment and are the only sources and sinks of data. Connectors are network components that connect devices and transport their data exhibiting certain configurable characteristics. Figure 1 shows a topology with four client devices, one server device, two switch connectors and one internet connector.

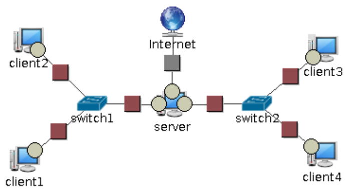


Fig. 1. Example Topology

3.1 Architecture

ToMaTo uses virtualization technologies to allow experiments to run concurrently in isolated environments spanning parts of the experimental facility. ToMaTo consists of three modules, the host system, the central back-end and the web-based front-end (http://tomato.german-lab.de) as shown in figure 2. The host system runs on all hosts of the experimental facility and offers virtualized resources to be controlled by the central back-end. The host hypervisor consists of a Linux operating system with the following additional components installed:

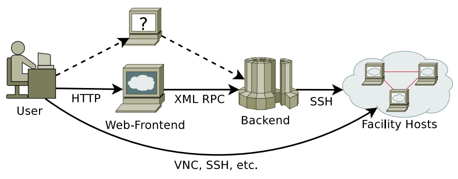


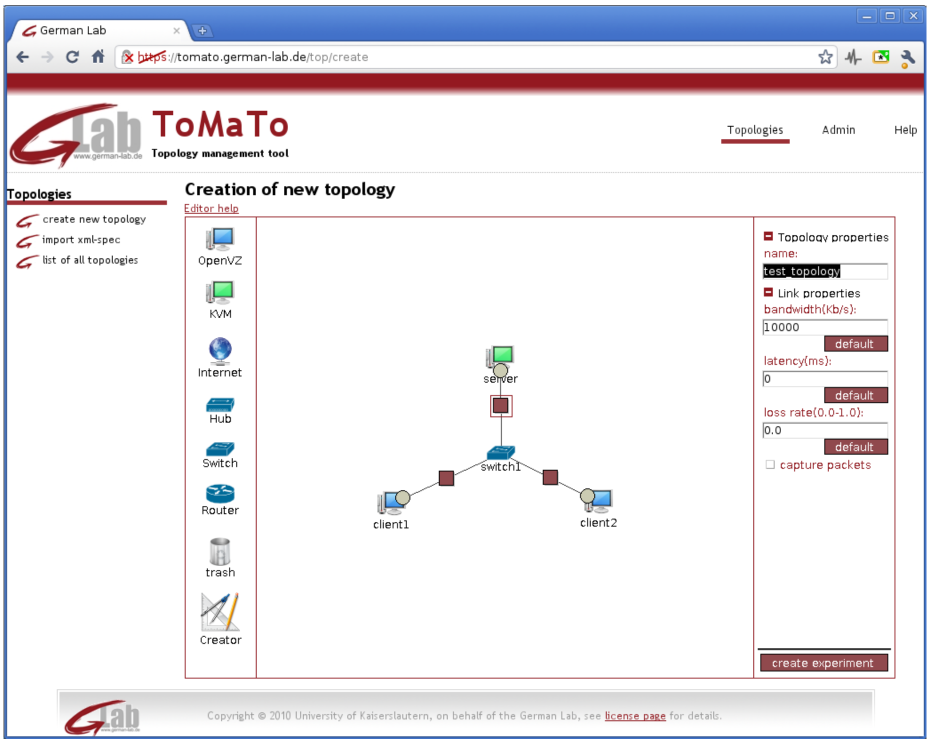
Fig. 2. ToMaTo structure

- PROXMOX VE[1] as virtualization tool for virtual machines
- Tinc[2] as virtualization tool for virtual networks
- Dummynet[6] as link emulation tool

[1] PROXMOX VE is a product of Proxmox Server Solutions GmbH, see http://pve.proxmox.com

[2] Tinc is a VPN software project by Tilburg university, see http://tinc-vpn.org

The host component allows the central back-end to configure and control these tools via a secure communication channel. This back-end is realized as a central component to allow easy resource management. It distributes the virtual machines evenly across the physical hosts to balance the load on those hosts. To keep the host environment as simple as possible the back-end contains all program logic and uses a secure communication channel to execute commands on the hosts. The back-end also manages user authentication and authorization and provides very basic accounting. Using an LDAP[3] server, existing user accounts can be easily integrated into the back-end.



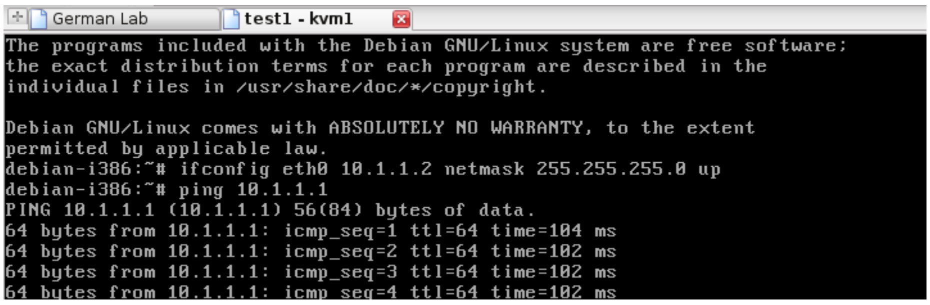
Topology properties
name: test_topology

Link properties
bandwidth(Kb/s): 10000 default
latency(ms): 0 default
loss rate(0.0-1.0): 0.0 default
☐ capture packets

Fig. 3. Graphical topology editor

The back-end offers an XML interface to be used by front-end tools. Currently only one front-end exists, but the API is generic enough to allow other front-ends as well. The main ToMaTo front-end consists of a website that allows users to create and edit their topologies using a graphical editor and to manage the topologies, devices and connectors. Figure 3 shows the ToMaTo website including the graphical editor, which is described in subsection 3.4.

[3] Lightweight Directory Access Protocol



```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
debian-i386:~# ifconfig eth0 10.1.1.2 netmask 255.255.255.0 up
debian-i386:~# ping 10.1.1.1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=64 time=104 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=64 time=102 ms
64 bytes from 10.1.1.1: icmp_seq=3 ttl=64 time=102 ms
64 bytes from 10.1.1.1: icmp_seq=4 ttl=64 time=102 ms
```

Fig. 4. VNC access to a KVM device

Users can also access their devices using a built-in VNC[4] viewer as shown in figure 4. Administrators can use a special section to access debugging information, manage the physical hosts and carry out various administrative tasks.

3.2 Device types & capabilities

ToMaTo currently supports two device types offered by PROXMOX: OpenVZ and KVM[5] virtual machines. OpenVZ is a container-based virtualization solution for Linux. OpenVZ virtual machines can run nearly all Linux systems and provide an environment similar to a physical machine from a user-space perspective. Since OpenVZ only runs a single kernel for all virtual machines some limitations arise. The kernel can not be modified, extended using modules or configured using *sysctl* from a virtual machine. Also the kernel version limits the guest systems to Linux operation systems that are compatible with a current kernel. Virtual network interfaces can be created on the OpenVZ host and exposed to the guest. Being a container-based virtualization, OpenVZ is very lightweight and offers flexible resource management (i.e. the external memory usage of a VM equals the usage inside the VM).

KVM offers a full virtualization including running one kernel per virtual machine and exposing emulated hardware to the VMs. Using KVM allows to run any x86-based operating system (even Windows and BSD) and to configure the operating system as needed. All hardware that is needed by the operating system including main board, hard disks and network interfaces is emulated by KVM. This offers maximal flexibility in choosing and configuring the VM but it also has a higher cost in terms of memory usage and performance reduction. ToMaTo offers both virtualization choices to the users so they can choose the optimal setups for their experiments. For both virtualization solutions ToMaTo offers pre-built virtual machines called templates. The users can choose between various Linux distributions in 32 and 64 bit architectures. For KVM also a pre-installed FreeBSD template is available. Users can also download and upload images of

[4] Virtual Network Computing
[5] Kernel-based Virtual Machine

their virtual machines. This can be used for backup purposes, to prepare an experiment first before actually running it or to build images containing a custom operating system that is not in the template list.

3.3 Connector types & capabilities

To connect the devices, and thus form a network topology, ToMaTo offers different options the user can choose from. The simplest option is the connector type "internet". This connector simply connects the network interface to the Internet. Network configuration is done automatically using DHCP. Using this connector, topologies can use external services, the user can access the exposed devices like servers over the Internet and even other testbed resources can be connected to ToMaTo topologies. The Internet connector does not allow any QoS[6] guarantees and, due to technical reasons, no QoS limitations can be set on this connector type. Other connector types use the Tinc VPN which connects devices in private networks that are not connected to each other nor to the Internet. Users can choose between hub, switch and router semantics in this private network. On connections using these connectors, network characteristics like packet loss, delay and bandwidth limitation can be emulated. Additionally users can capture network traffic at these connectors and download it as a *pcap* file to analyze or view the traffic using specialized tools like Wireshark[7].

3.4 Graphical editor & topology creator

To create topologies consisting of the presented devices and connectors, a graphical editor, see figure 3, has been included in the web-based front-end. The editor provides a simple drag and drop interface for topology creation as well as configuration menus for all properties of the single network components. Components can be added to the topology by selecting them from the panel on the left hand side of the editor and dragging them to the working area. Components in the topology can be connected to each other by holding down the Ctrl-key and subsequently selecting all icons that should be connected. For instance, two hosts can be connected by connecting both of them to an intermediate hub, switch, or router. The right hand side of the graphical editor provides a property menu where the properties of the currently selected component, interface or link can be configured. Depending on the selected entity, these properties include IP-addresses, host names, network characteristics, and so on.

The graphical editor described so far is enough to create topologies of any size and complexity. However, if large topologies have to be created, a lot of manual configuration work is necessary. Furthermore, from an illustration point of view, if a network consists of more than a few nodes, it becomes difficult to arrange the icons inside the working area in a way so that the topology structure is still clearly visible. Therefore, the graphical editor has been equipped with an

[6] Quality of service

[7] Wireshark is a free and open-source packet analyzer, see http://www.wireshark.org

automatic topology creator (TC). The basic idea of TC is to take as much manual work as possible from the user and to automate it. TC allows automatic creation of a previously configured number of identical hosts, to arrange these hosts in an ordered way inside the working area and to connect these nodes to each other in a predefined topology structure and with automatically assigned IP-addresses and host names. So far, TC can connect the hosts either in a star topology with a central switch or host, in a ring, or fully-meshed, i.e. each host connected to each of the other hosts. Depending on the selected option, the IP-address subnets and netmasks are automatically configured and the IP-addresses are assigned to the host interfaces subsequently.

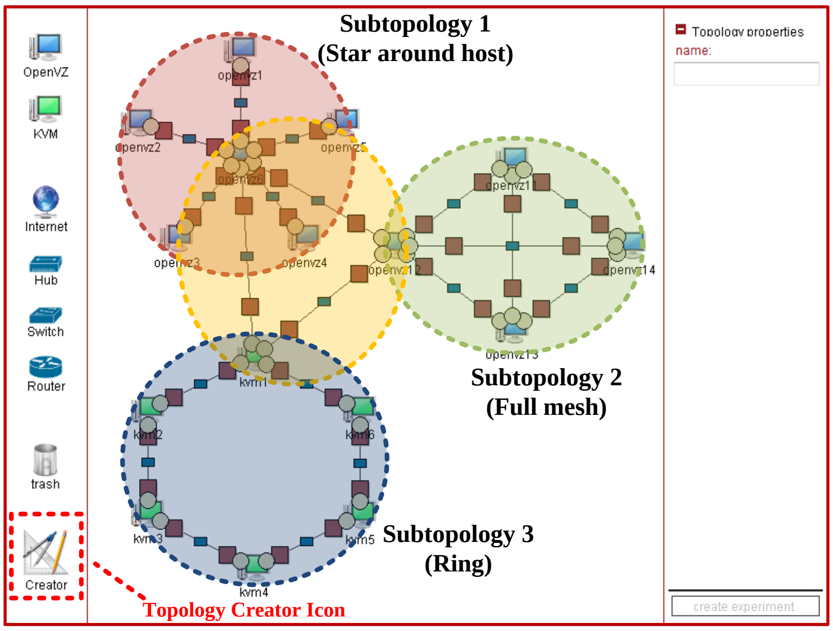


Fig. 5. Topology creator - Screenshot with example topology and illustration of structure

With TC, it is not only possible to create new hosts and connect them, but to connect (and auto-configure) already existing nodes in the network. Figure 5 illustrates the benefits of this functionality both in a general structure and in a concrete example topology. TC allows easy creation of any kind of hierarchically structured topology. The procedure starts bottom-up by creating all of the subtopologies (subtopology 1, 2, . . . , i, . . . , n) with desired node counts, connection types and IP-subranges and place them in the working area. Then (gateway) hosts of the different subtopologies are selected and connected to each other to create a topology on top of the subtopologies. These steps can be repeated arbitrarily often to design a final topology of several hierarchical layers.

In the example displayed in figure 5, a 5-host star topology around a sixth host, a 4-host full mesh, and a 6-host ring are connected to each other by selecting one host of each subtopology and fully meshing these hosts.

4 Evaluation

Evaluating a software design 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 experimental facility software the design goal is to support experiments and help researchers carry out their experiments. To evaluate ToMaTo based on this goal section 4.1 first develops a classification of experiments and section 4.2 outlines how ToMaTo supports these types of experiments. Section 4.3 takes a quick look at the efficiency and scalability of ToMaTo.

4.1 Types of experiments

The following experiment types have been identified in the German-Lab project.

Access layer experiments consider the lower networking layers and examine the usage of hardware for networking. An example for this experiment class are 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, Wifi, Fiber, etc.). In most cases, these requirements can only be fulfilled with custom testbeds, so supporting this kind of experiment was not a design goal for ToMaTo.

Network layer 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 work on top of the network layer and consider protocols and algorithms for bigger networks. Nearly all peer-to-peer experiments fall in this category. These experiments need a high number of devices but not much hardware access, especially no kernel access. They only need simple network topologies with link emulation.

Legacy application experiments contain legacy software, i.e. widespread software that 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[9] 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 requirements of the classes are very heterogeneous but a general trade-off between more resource access and access to more resources becomes evident.

4.2 Experiment support in ToMaTo

ToMaTo has been designed to support all experiment classes identified in section 4.1 except for **access layer experiments** because these experiments need a specialized experimental facility depending on the access technology. The Wisebed and DES testbeds for example are specialized experimental facilities for sensor networks and wifi.

Network layer experiments can be done easily in ToMaTo using KVM devices and switch connectors. The KVM devices offer all needed flexibility in kernel choice and modification required by this experiment class. Switched networks are layer-3-agnostic so any TCP/IP modification or substitute can be examined. Using the graphical editor 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 and switch or router connectors. Since OpenVZ devices are very lightweight, a high number of devices can be used in topologies. Using an Internet connector, external resources like Planet-Lab nodes can be included in the experiment. The topology creator makes it very easy to design huge experiments with ring or star topologies. Using the upload/download image feature, users can prepare a device image once and upload it to all of their devices. Capturing network traffic 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, so 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.

4.3 Efficiency and scalability

With ToMaTo, users can choose between OpenVZ and KVM virtualization. This way users can get the level of access that is needed for their experiments and still use as few resources as possible. A modern cluster node can handle up to 250 OpenVZ devices and up to 50 KVM devices, both depending on device usage. The connector components only pose a very small overhead and can handle connections with over 100 Mbps.

ToMaTo hosts use an existing operating system as basis and only need small changes that have been bundled as software package. That means that support and security updates are available 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.

5 Conclusion

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 experimental facility 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 experimental facility currently 20 of 182 hosts are ToMaTo-enabled. The goal is to increase this number to about 50 and thereby increase the usability of the testbed. Early plans exist to integrate support for Openflow hardware and software to allow even complexer network topologies.

References

1. *DES-Testbed A Wireless Multi-Hop Network Testbed for future mobile networks*, Stuttgart, Germany, 06/2010 2010.
2. J. Ahrenholz, C. Danilov, T. Henderson, and J.H. Kim. Core: A real-time network emulator. In *Proceedings of IEEE MILCOM Conference*, 2008.
3. 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.
4. Y. Benchaïb and A. Hecker. Virconel: A new emulation environment for experiments with networked it systems. In *High Performance Computing & Simulation Conference*, 2008.
5. 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.
6. Marta Carbone and Luigi Rizzo. Dummynet revisited. *Computer Communication Review*, 40(2):12–20, 2010.
7. Paul Müller and Bernd Reuther. Future internet architecture - a service oriented approach. *it - Information Technology*, 50(6):383–389, 2008.
8. Larry L. Peterson, Andy C. Bavier, Marc E. Fiuczynski, and Steve Muir. Experiences building planetlab. In *OSDI*, pages 351–366. USENIX Association, 2006.
9. 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.
10. Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI*, 2002.