An Overview of Linux Container Based Network Emulation

Schalk Peach[1, 2], Barry Irwin[2] and Renier van Heerden[1]
[1]Council for Scientific and Industrial Research, South Africa
[2]Rhodes University, Grahamstown, South Africa
speach@csir.co.za
b.irwin@rhodes.ac.za
rvheerden@csir.co.za

Abstract: The objective of this paper is to assess the current state of Container-Based Emulator implementations on the Linux platform. Through a narrative overview, a selection of open source Container-Based emulators are analysed to collect information regarding the technologies used to construct them to assess the current state of this emerging technology. Container-Based Emulators allows the creation of small emulated networks on commodity hardware through the use of kernel level virtualization techniques, also referred to as containerisation. Container-Based Emulators act as a management tool to control containers and the applications that execute within them. The ability of Container Based Emulators to create repeatable and controllable test networks makes it ideal for use as training and experimentation tools in the information security and network management fields. Due to the ease of use and low hardware requirements, the tools present a low cost alternative to other forms of network experimentation platforms. Through a review of current literature and source code, the current state of Container-Based Emulators is assessed. The primary source of information is publications by the creators of the selected Container-Based Emulators. Each Container-Based Emulator is introduced with a brief summary of the history and requirements that lead to its creation. The reader is presented with a comparison of the specific kernel level virtualization technologies used to implement the virtualization sub-system of the Container-Based Emulator. The structural design of Container-Based Emulators is analysed and summarized to provide a concise view of the capabilities that users are presented with. An architectural model is introduced that can assist with the selection of a Container-Based Emulator, based on the requirements of the end user. The paper concludes with a summary of the current state of Container-Based Emulators, and proposes future research areas to be explored.

Keywords: Linux, containerisation, network emulation, container-based emulator

1. Introduction

Operating system level virtualisation, also known as containers, is applied in various circumstances such as application packaging and low overhead virtual private servers (Merkel, 2014; Soltesz, Pötzl, Fiuczynski, Bavier, and Peterson, 2007). When combined, Linux containers and network virtualisation techniques such as Linux bridges provides the base technologies to create low overhead virtual networks on commodity hardware (Bhatia, Motiwala, Muhlbauer, Mundada, Valancius, Bavier, Feamster, Peterson, and Rexford, 2008; Lantz, Heller, and McKeown, 2010). These software applications are often referred to as Container Based Emulators (CBE). CBEs provide a low cost, low maintenance alternative to other forms of network experimentation platforms. These systems can be deployed as sandboxed training and experimentation platforms for various fields including network administration and information security. The aim of this paper is to collect the current knowledge regarding software applications specialising in providing a frontend for the creation of the virtualised computer networks. A narrative overview methodology is used to assess the current state of open source Linux container based network emulation through a study of literature published on the subject matter. The reader is presented with the information required to select a CBE that is suitable to the task at hand.

1.1 Related work

Linkletter, 2015 provides reviews of most of the Container-Based Emulators discussed in this paper. The reviews provided are however focused on installation and usage. In contrast, this paper focuses on technical aspects of Container-Based Emulators. In (Pizzonia and Rimondini, 2014), a collection of network simulation and emulation platforms are compared based on the authors requirements for deployment in an educational context. The comparison subsequently focuses on deployment, ease of use and technology support, in preference to technical aspects of these systems. Salopek (Salopek, Vasic, Zec, Mikuc, Vasarevic, and Koncar, 2014) compares the overhead incurred in container-based emulators based on the method of node virtualisation.

1.2 Structure

In Section 2 a brief introduction to different types of network experimentation platforms is given, highlighting the advantages and disadvantages of CBEs. In Section 3 current open source CBEs are introduced. In Section 4,

the selected CBEs are compared at technology and feature level. Finally, in Section 5, conclusions on the current state of CBEs are presented, as well as highlighting alternative methods used to create network experiments using operating system level virtualisation.

2. Network experimentation platforms

Network experimentation platforms enable researchers to execute repeatable experiments, ensuring consistent results. These platforms can be constructed using various techniques. This section considers four broad technology areas that can be employed to create network experimentation platforms and gives a brief overview of each type.

2.1 Testbeds

A testbed is a deployment of computer and networking hardware that aims to replicate the conditions in which a software application will be utilised. Key goals of a testbed is to recreate expected conditions at the highest possible level of fidelity. One of the advantages of a testbed is absolute fidelity of the components used to construct the experimentation platform, resulting in repeatable tests. Due to the large number of hardware components and space required, the cost of deploying a testbed can be prohibitive.

2.2 Virtualisation

The cost of deploying a testbed can be reduced through the use of virtualisation. The functional fidelity of virtualised computer hardware is near perfect, complementing the repeatability of experiments. By replacing costly end user hardware with virtualised instances, the total hardware required is reduced. Additional advantages of virtualisation are reductions in physical space and maintenance requirements.

2.3 Simulation

An alternative to testbeds, with or without virtualisation, is simulation. Through simulation, hundreds to thousands of nodes in a computer network can be simulated on a single high end server. The behaviour of simulated components is restricted to the accuracy of the models employed to simulate the component. This could result in a loss of accuracy if any non-deterministic (within the scope of modelling) component is included in the simulation.

2.4 Emulation

The introduction of operating-system level virtualisation (containerisation) link emulation tools in Linux, FreeBSD and Solaris introduces the possibility to create experimental networks using operating system components. Containers have little overhead and provide a range of isolation methods to assist in the creation of experimental networks. Container-Based Emulators exploit these components to provide the user with a convenient environment to create network configurations. The use of containerisation has some disadvantages as there is a marked loss of fidelity in network traffic throughput and metrics as node density is increased. Pressure on random access memory could result in adverse effects during experiment with high node count. Systems using these techniques are often referred to as emulation systems.

3. Container-Based emulator implementations

Within the context of this paper, a Container-Based Emulator is defined as a purpose made suite of utilities and applications that abstracts the complexity of creating networked containers and enables a user to define and instantiate a set of networked containers, with the ability to define and control configuration values of each deployed component, through a single interface. In this section a set of open-source Container-Based Emulators are reviewed and a brief overview of each CBE is given with regards to history, goals and construction.

3.1 Mininet

MiniNet (Lantz, Heller, and McKeown, 2010) started out as a project to enable large scale OpenFlow (McKeown, Anderson, Balakrishnan, Parulkar, Peterson, Rexford, Shenker, and Turner, 2008) experimentation on commodity hardware. The MiniNet project was then expanded to increase functional realism of network simulations, which resulted in Mininet-HiFi (Handigol, Heller, Jeyakumar, Lantz, and McKeown, 2012; Heller, 2013). The MiniNet GUI exposes a minimal set of components to construct a network topology. The base

components, a host, an openflow switch and controller, a basic switch and basic router is provided. The GUI offers options to provide minimal configuration of each component. Constructing a network topology using the command line tools and configuration files allows the user to exert greater control over network topology and component configuration.

3.2 IMUNES

The Integrated Multiprotocol Network Emulator/Simulator (IMUNES) (Zec and Mikuc, 2004; Salopek, Vasic, Zec, Mikuc, Vasarevic, and Koncar, 2014) is actively developed by the University of Zagreb. Development of the concepts used IMUNES can be traced back to 2002 to a project to virtualise the BSD network stack (Zec, 2002). The goal of IMUNES is to be an alternative to computer network testbeds that can be used on commodity hardware. The IMUNES GUI is developed as an extensive network topology configuration tool. The default component provided include basic networking components such as a hub, switch, router, a PC and a host, as well as a click router and switch. Each component provided can be extensively configured through the user interface.

3.3 Common open research emulator

The Common Open Research Emulator (CORE) (Ahrenholz, 2010) started off as a fork of IMUNES by the United States Naval Research Laboratory (NRL) and Boeing, and is maintained by the Networks and Communication Systems Branch of the NRL. The CORE project extended IMUNES with the ability to execute on Linux, a remote procedure call (RPC) application programming interface (API), a Python library and various user interface (UI) enhancements. Additional goals of the CORE project are to allow wireless network experiments through the Extendable Mobile Ad-hoc Network Emulator (EMANE) (Ahrenholz, Goff, and Adamson, 2011), and the ability to distribute network emulation across multiple hosts. The components provided through the CORE GUI is arranged into two different sets, Layer 2 and Layer 3 components. Layer 2 components include a hub, switch, a wireless LAN emulator and physical Ethernet connections. Layer 3 nodes include a host, PC, a router and a MANER Designated Router (MDR). Layer 3 nodes in CORE can be configured to execute pre-defined service as well as user defined services.

3.4 NetKit and NetKit-NG

NetKit (Pizzonia and Rimondini, 2008; Pizzonia and Rimondini, 2014; Rimondini, 2007) is a project by the Computer Networks Laboratory of the Roma Tre University to enable network experiments to be executed on commodity hardware. NetKit-NG (Iguchi-Cartigny, 2014) is a fork of Netkit, aiming to update the operating system version used. NetKit and NetKit-NG does not provide a GUI, however 3rd party tools such as Visual Netkit Fazio and Minasi, 2009 are available.

3.5 VNX and VNUML

Virtual Networks over LinuX (VNX) (Fernandez, Cordero, Somavilla, Rodriguez, Corchero, Tarrafeta, and Galan, 2011) is a continuation of the Virtual Network User Mode Linux (VNUML) project (Galan, Fernandez, Ruiz, Walid, and Miguel, 2004). VNUML started as an emulation platform to study the address assignment model in IPv6 (Fernandez, De Miguel, and Galan, 2004). The emulation platform used for the study was developed into VNUML to support research project related to computer networks. Development of the VNUML platform was halted in 2009 and has been replaced by VNX. The goals of VNX is to include virtualisation tools to support operating systems other than the host platform in network experiments. It incorporates libvirt and DynaMIPS to achieve these goals. VNX does not have a graphical user interface, however it can produce a graphical map of the current emulation.

3.6 Topology management tool

The Topology Management Tool (ToMaTo) (Schwerdel, Hock, Günther, Reuther, Müller, and Tran-Gia, 2012) is developed by a multi-party group as part of the German-Lab. The goal of ToMaTo is to be a general network experiment environment. Network components in ToMaTo are organised into Devices and Connectors. Devices in ToMaTo emulate networked machines that send and receive packets. Connectors in ToMaTo emulate the links that connect devices and emulate link characteristics.

3.7 Marionnet

The Marionnet project (Loddo and Saiu, 2007; Loddo and Saiu, 2008) was developed by Jean-Vincent Loddo as a teaching aid for his course in networking at the Universit Paris 13. Network components in Marionnet is organised into virtual computers and virtual network devices. The virtual computer components emulate networked machines on the emulated network and virtual network devices emulates hubs, switches, routers and links in the emulated network. A virtual external socket component is provided to link physical Ethernet ports on the host machine to the emulated network. Marionnet provides a desktop application user interface that allows the end user to configure each component in detail.

3.8 Cloonix

The Cloonix network emulation tool (Rehunathan, Bhatti, Perrier, and Hui, 2011; Perrier, 2015) was created by V. Perrier as a tool to assist in the automated creation of simulated network. In contrast to CBE's, Cloonix exclusively uses KVM to virtualise computers, with no option to utilise containerisation techniques. Cloonix has the same objectives as CBE's, a network experimentation platform capable of running on a single host machine, and is thus included. Network components in Cloonix are organised into lan, kvm, tap, c2c and snf objects. The lan object emulates network equipment such as hubs, switches and Ethernet connections, with the kvm component being responsible for creating guest machines in the instantiated topology. The tap, c2c and snf tools provide additional features to connect to physical Ethernet ports on the host, Cloonix to Cloonix distributed emulation and packet sniffing, respectively. The Cloonix user interface provides the user with basic network topology creation tools. Creating a network topology using configuration allows fine grained control over each component in the network topology and allows scripts to be executed on each component during start-up.

4. Container-Based emulator comparison

Each of the CBEs that forms part of this study will be compared in detail using two different comparison methods. The first part will take a look at the architectural choices made to construct the CBE. The second part will compare the specific technologies used to implement the CBE framework.

4.1 Architecture

The architectural choices made during the implementation of each CBE is analysed and compared to better understand the current state of CBEs as frameworks for network experimentation. Each CBE is analysed to assess choices regarding the human-machine interface, how it exposes backend functionality, the design of the backend and the choice of virtualisation technologies. An additional comparison that is included is the capability of a CBE to distribute an emulation across multiple host machines. In Table 1, a preliminary model is shown that will be used to analyse each CBE.

User Interface Each CBE is compared on the interface that it provides to the end user. CBEs that expose only command line (CLI) based interfaces are more difficult to use for beginners but could provide more control for experienced users, whereas CBEs that expose graphical user interfaces (GUI) lower the entry barrier, making them usable for a wider audience. Some CBE implementations expose both GUI and CLI capabilities.

Remote Control The architecture of each CBE is analysed to assess the capabilities of the backend component. The remote control level determines if the CBE framework exposes a remote procedure call (RPC) API, resulting in a loosely coupled architecture. This type of architecture will most commonly expose a backend API library that can be integrated into a CLI and enable distributed emulation. The alternative is a monolithic design, where the exposed user interface (either GUI or CLI) directly controls instantiation of the emulation network.

Virtualisation The comparison of CBE virtualisation techniques assesses the type of technology used to instantiate nodes within the network topology. Each CBE implementation can either use only containerisation or use containerisation and virtualisation. An outlier is Cloonix, which uses full virtualisation, it is included as it is built for the same purpose as CBEs, although it uses only virtualisation technology (KVM).

Distributed Emulation CBEs are compared with respect to the capability of the emulation to be distributed across multiple host machines. Distributed emulation enables the emulation capacity to be expanded horizontally. Distributed emulation capability of a CBE is dependent on the remote control and backend implementation.

Through an analysis of the literature and source code of the CBEs, it was determined that the software packages reviewed shared common architectural designs. In Table 1 the results of the review are shown.

Table 1: Container-Based emulator architecture comparison

Implementation	User Interface	Remote Control	Virtualisation Library	Virtualisation Technology	Distributed
CORE	GUI, CLI	Binary RPC	API Library	Container and Virtualisation	Yes
IMUNES	GUI	None	Monolithic	Container	None
Mininet	GUI, CLI	None	API Library	Container	None
VNX	CLI	3rd Party	API Library	Container and Virtualisation	3rd Party
ToMaTo	GUI	XML-RPC	API Library	Container and Virtualisation	Yes
Marionnet	GUI, CLI	None	Monolithic	Container	Multi-instancing
NetKit	GUI, CLI	None	API Library	Container	Multi-instancing
Cloonix	GUI, CLI	Remore Access	API Library	Virtualisation	Yes

From the tabled results, each of the sections of the architectural model (Table 1) can be subdivided into common approaches used. The user interface component for the all but one of the CBEs is based on a graphical user interface (GUI). VNX does not by default have a user interface, but can create a graphical map of the emulated network through ImageMagick. Half of the reviewed CBEs integrate remote control functionality to allow for distributed emulation. Marrionnet and NetKit achieve distributed emulation through multi-instancing. Only two of the reviewed CBEs (IMUNES and Marionnet) are based on a monolithic design with all other CBEs opting for a component based architecture. CORE, VNX and ToMaTo utilise the strengths of both virtualisation and containerisation to allow for more control of nodes within the experiment if required.

The comparison model of Table 1 is expanded in Table 1 to differentiate between single instance and distributed CBEs.

Table 2: Container-Based emulator architecture model

	CBE Feature	Distributed	Single Instance
System Abstraction	User Interface	Graphical User Interface, Command Line Interface	
	Remote Control	Remote Control Daemon	-
	Virtualisation Library	Application Programming Interface	Monolithic Application, Application Programming Interface
	Virtualisation Technology	Node Emulation, Network Emulation, Link Emulation	

CBEs capable of distributed emulation favours and API model for interacting with containerisation and virtualisation components of the host operating system, though the most significant difference is that single instance CBEs lack remote control capability.

4.2 Implementation

In this section, the technologies that are used to implement the main features of a CBE are enumerated. A CBE has to address a minimum of two aspects, nodes and topology, the base requirements for a computer network. A third aspect of a computer network, link metrics, is addressed by some CBEs. Built in capability to generate background traffic in the emulated network is not addressed in this paper.

Node Emulation The first requirement that a CBE needs to address is that of virtualised nodes within the emulated network topology. Each CBE is analysed with respect to the different technologies that it can use to instantiate nodes. The technologies used to instantiated nodes can range from existing well known systems such as UML, to custom containerisation implementations to address needs specific to the CBE.

Network Emulation The second requirement that a CBE needs to address is the creation of a network topology that links the instantiated nodes. Each CBE is again analysed to determine the technologies used to emulate a

network topology. The network topology instantiated requires a virtual network interface card (NIC) mounted in the emulated node, and a method to connect these NICs to each other, or a virtual switch.

Link Emulation The third component of implementation comparison is link emulation. Link emulation addresses the need to control the characteristics of network traffic flowing in the instantiated topology. The ability to control link metrics such as throughput and packet loss increases the realism (fidelity) of the emulated network, allowing replication of real world conditions for network experiments.

Table 3 lists the version of the CBE that was reviewed, it's host operating system and the different technologies used to implement the CBE.

Table 3: Container-Based emulator technology comparison

Implementation	Version	Operating System	Node Emulation	Network Emulation	Link Emulation
CORE	4.8	Linux	namespaces, LXC, xen	brctl	ebtables
CORE	4.8	FreeBSD	jails	netgraph, vnet	netgraph pipe
IMUNES	2.1.0	FreeBSD	jails	netgraph, vnet	netgraph pipe
IMUNES	2.1.0	Linux	Docker	ovs	
Mininet	2.2.1	Linux	cgroup, netns	ovs, ivs, openflow, brctl	tc, netem
VNX	2.0	Linux	dynamips, libvirt, lxc, uml, vbox, netns	uml switch, ovs, brctl	tc
ToMaTo	3.5.3	Linux	openvz, kvm	tinc vpn, brctl	ipfw (dummynet)
Marionnet	0.90.6 (457)	Linux	uml	vde switch, brctl, tunctl	vde plug
NetKit	2.8	Linux	uml	uml switch	
Cloonix	26.02	Linux	kvm	uml cloonix switch, cloonix mulan	t2t (currently deprecated)

The FreeBSD capable CBEs both use jails and the netgraph system to create emulated networks. Linux is the most popular host operating system for CBEs as all CBEs run on Linux.

UML (User Mode Linux) and KVM (Kernel-based Virtual Machine) is a popular choice as a virtualisation backend with 5 of the 8 CBEs using these technologies. Recent container management systems such as LXC and Docker is by CORE and IMUNES respectively. The only CBE not relying on a 3rd party application to manage containerisation is Mininet. The design goals of Mininet prompted the developers to create a custom container management system to exercise better control over resource usage.

Network emulation on Linux is largely based on Linux bridges (brctl). All CBEs but CORE include support for 3rd party network emulation applications, in particular Open vSwitch. Cloonix is the only CBE to implement custom network emulation technologies. Link emulation is not supported in all CBEs and there is no clear favourite technology for implementing this feature.

5. Conclusion

Available network experimentation platforms give the end user a choice of fidelity level that is most suited to experiments that are to be done. CBEs as an alternative network experimentation platform presents a middle ground in terms of node density and fidelity. CBEs have the ability to have hundreds of nodes in an experimental network while still having access to an operating system kernel capable of executing real world applications. This allows for experimentation that requires interaction with real world applications at a large scale. The open source CBEs reviewed vary in architecture and implementation. These variations in architecture and implementation specifics of the different CBEs reviewed allows the end user to select the most appropriate system based on his or her requirements. For education and training environments, the availability of a graphical user interface supersedes the ability to programmatically control the experimental network. In contrast, experimentation with large scale networks that span multiple computers will benefit from the ability to exercise remote programmatic control over the experimental network. CBEs present a viable, low cost alternative for network administration, education, and security specialists. The specific requirements of an experimental setup will lead the end user to select a CBE that can function within constraints of the environment that the experiment will be executed.

References

Ahrenholz, Jeff (2010). “Comparison of core network emulation platforms”. In: *Military Communications Conference, 2010-MILCOM 2010*. IEEE, pp. 166–171.

Ahrenholz, Jeff, Tom Goff, and Brian Adamson (2011). “Integration of the core and emane network emulators”. In: *Military Communications Conference, 2011-MILCOM 2011*. IEEE, pp. 1870–1875.

Bhatia, Sapan et al. (2008). “Trellis: A platform for building flexible, fast virtual networks on commodity hardware”. In: *Proceedings of the 2008 ACM CoNEXT Conference*. ACM, p. 72.

Fazio, Alessio Di and Paolo Minasi (2009). authors found at http://list.dia.uniroma3.it/pipermail/netkit.users/2008-July/000368.html. url: https://code.google.com/archive/p/visual-netkit/ (visited on 03/08/2016).

Fern´andez, David et al. (2011). “Distributed virtual scenarios over multi-host Linux environments”. In: *Systems and Virtualization Management (SVM), 2011 5th International DMTF Academic Alliance Workshop on*. IEEE, pp. 1–8.

Fern´andez, David, Tom´as De Miguel, and Ferm´ın Gal´an (2004). “Study and emulation of IPv6 Internetexchange-based addressing models”. In: *Communications Magazine, IEEE* 42.1, pp. 105–112.

Galan, F. et al. (2004). “Use of virtualization tools in computer network laboratories”. In: *Information Technology Based Higher Education and Training, 2004. ITHET 2004. Proceedings of the Fifth International Conference on*, pp. 209–214. doi: 10.1109/ITHET.2004.1358165.

Handigol, Nikhil et al. (2012). “Reproducible network experiments using container-based emulation”. In: *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, pp. 253–264.

Heller, Brandon (2013). “Reproducible Network Research with High-fidelity Emulation”. PhD thesis. Stanford University.

Iguchi-Cartigny, Julien (2014). *NetKit-NG*. url: http://netkit-ng.github.io/ (visited on 03/08/2016).

Lantz, Bob, Brandon Heller, and Nick McKeown (2010). “A network in a laptop: rapid prototyping for software-defined networks”. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, p. 19.

Linkletter, Brian (2015). *Open-Source Routing and Network Simulation*. url: http://www.brianlinkletter. com/open-source-network-simulators/ (visited on 06/18/2015).

Loddo, Jean-Vincent and Luca Saiu (2008). “Marionnet: a virtual network laboratory and simulation tool”. In: *First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems*.

Loddo, Jean-Vincent and Luca Saiu (2007). “Status report: marionnet or how to implement a virtual network laboratory in six months and be happy”. In: *Proceedings of the 2007 workshop on Workshop on ML*. ACM, pp. 59–70.

McKeown, Nick et al. (2008). “OpenFlow: enabling innovation in campus networks”. In: *ACM SIGCOMM Computer Communication Review* 38.2, pp. 69–74.

Merkel, Dirk (2014). “Docker: lightweight linux containers for consistent development and deployment”. In: *Linux Journal* 2014.239, p. 2.

Perrier, V (2015). *Cloonix*. url: http://clownix.net (visited on 08/02/2015).

Pizzonia, Maurizio and Massimo Rimondini (2008). “Netkit: easy emulation of complex networks on inexpensive hardware”. In: *Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), p. 7.

Pizzonia, Maurizio and Massimo Rimondini (2014). “Netkit: network emulation for education”. In: *Software: Practice and Experience*.

Rehunathan, D et al. (2011). “The study of mobile network protocols with virtual machines”. In: *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 115–124.

Rimondini, Massimo (2007). “Emulation of computer networks with Netkit”. In: *Dipartimento di Informatica e Automazione, Roma Tre University, http://www. netkit. org/, RT-DIA-113-2007*.

Salopek, Denis et al. (2014). “A network testbed for commercial telecommunications product testing”. In: *22nd International Conference on Software, Telecommunications and Computer Networks-SoftCOM 2014*.

Schwerdel, Dennis et al. (2012). “ToMaTo-a network experimentation tool”. In: *Testbeds and Research Infrastructure. Development of Networks and Communities*. Springer, pp. 1–10.

Soltesz, Stephen et al. (2007). “Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors”. In: *ACM SIGOPS Operating Systems Review*. Vol. 41. 3. ACM, pp. 275– 287.

Zec, Marko (2002). “BSD Network stack virtualization”. In: *BSDCon Europe, Amsterdam, Nov* 2.

Zec, Marko and Miljenko Mikuc (2004). “Operating system support for integrated network emulation in imunes”. In: *1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)*, pp. 3–12.