Virtualbricks for DTN Satellite Communications Research and Education

Pietrofrancesco Apollonio[1], Carlo Caini[1(✉)], Marco Giusti[1], and Daniele Lacamera[2]

[1] DEI-ARCES, University of Bologna, Bologna, Italy
f.apollonio@ldlabs.org, carlo.caini@unibo.it, marco.giusti3@studio.unibo.it
[2] TASS Technology Solutions, Leuven, Belgium
daniele.lacamera@tass.be

Abstract. Virtualbricks is a virtualization solution for GNU/Linux platforms developed by the authors and included in Debian. The paper aims to show its potential, referring to version 1.0, just released, when applied to both research and education on DTN satellite communications. In brief, Virtualbricks is a frontend for the management of Qemu/KVM Virtual Machines (VMs) and VDE virtualized network devices (switches, channel emulators, etc.). It can be used to manage either isolated VMs, or testbeds consisting of many VMs interconnected by VDE elements. Among the wide variety of possible applications, with or without VM interconnections, the focus here is on the development of a virtual testbed on DTN satellite communications, a task for which Virtualbricks was especially designed. After having introduced the main characteristics of Virtualbricks, in the paper we will show how to set-up a Virtualbricks testbed, taking as an example a testbed recently used by the authors to investigate Moon communications through orbiters. The validity of Virtualbricks results is confirmed by comparison with results achieved on a real testbed, set-up for this purpose. The same testbed has also been successfully used for educational purposes at the University of Bologna.

Keywords: Testbed virtualization · DTN · Satellite communications · KVM · Qemu · VDE

1 Introduction

Several complete solutions for virtualization management have been developed to simplify design, configuration and management of Virtual Machines (VMs). Currently available virtualization management tools include proprietary suites like VMware® [1], Microsoft® SCVMM [2], Paragon® VM [3], SolarWinds® [4] and many more. Other solutions are based on Free or Open Source software and in particular on the use of the Kernel based Virtual Machine (KVM) [5], which is now part of the Linux kernel. Unfortunately, none of these solutions focuses on the design and management of complex network layouts, as they were created for cloud management and the virtualization of enterprise server farms does not usually require complex topologies.

I. Bisio (Ed.): PSATS 2014, LNICST 148, pp. 76–88, 2016.
DOI: 10.1007/978-3-319-47081-8_7

There are few examples in the literature that exploit virtualization technologies to provide virtual equivalent of testbed components, like routers and switches. The most advanced is possibly provided by Marionnet [6, 7]. Marionnet uses Virtual Distributed Ethernet (VDE) [8, 9], to emulate the network infrastructure needed to interconnect the VMs in complex networks. Unfortunately, Marionnet is mostly intended for educational purposes and the design choice to implement VMs via User-mode Linux (UML) [10] greatly penalizes performance. A different approach to build network testbeds is followed by Mininet [11], which allows the user to run a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. It is however a powerful network emulator, not a VM manager. In particular, its nodes do not run independent OSs and must share the file system of the host.

Since we found no existing solutions that would meet our requirements, we started the development of Virtualbricks [12], with the aim of creating a virtualization management tool focused on network design. Virtualbricks relies on Qemu [13], KVM and VDE; it is released under the GNU General Public License version 2 and has been included in the GNU/Linux Debian distribution. The paper refers to the version 1.0, just released, which implied an in-depth redesign of many part of the code to improve stability and add new significant features. Virtualbricks stems from the fusion of Qemulator, a front-end for Qemu, and Virtual NetManager, a project previously developed by the authors. The aim of the project is to provide an easy interface not only for VM management, but also for virtual network design. Although not exclusive, one of the most significant applications is the virtualization of real testbeds, such as those used by the authors [14] to evaluate TCP and DTN (Delay-/Disruption- Tolerant Networking) protocols in challenged networks [15]. As well as for research, Virtualbricks has a great potential also as an educational tool, as it allows students to build a complete networking testbed on their own PCs. Many new features introduced in the release 1.0 derive from the practical experience gained by the use of release1.0 beta versions in the LAB activities of a TLC Engineering Master course of the University of Bologna [16].

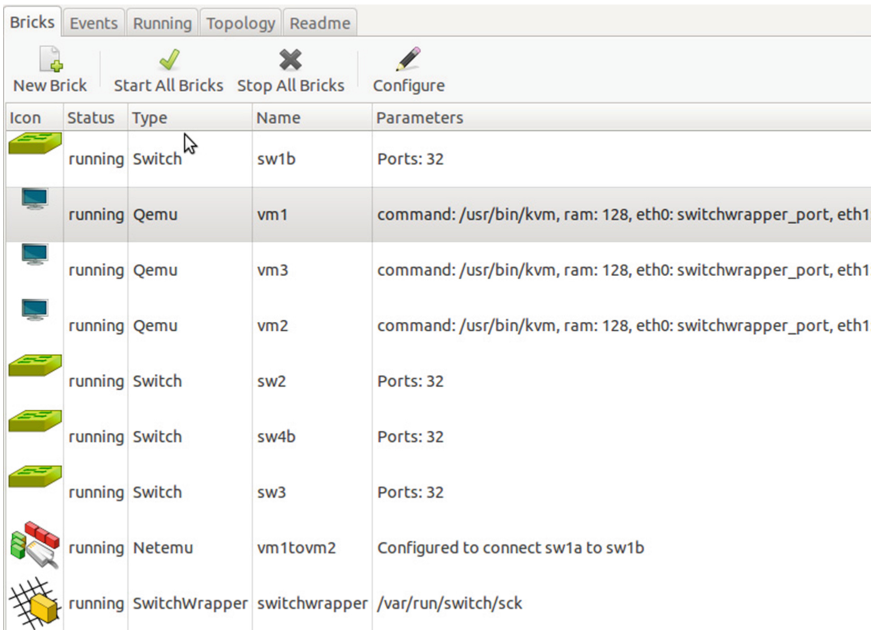
This paper aims to present the main features of Virtualbricks seen from the user point of view. Although many applications do not require VM interconnections, here the focus is on the development of a virtual testbed for scientific research on satellite DTN communications, a task for which Virtualbricks was especially designed. To this end, after a general description of Virtualbricks, the steps necessary to build the virtual testbed that was used by the authors in [17] are presented. The accuracy of Virtualbricks results will be then validated by comparisons with those achievable with a real testbed, expressly built up for this purpose. It is however worth stressing that Virtualbricks is a front-end, thus performance achievable (in terms of accuracy and speed) is the same as that achievable by its components, KVM or Qemu for VMs, and VDE for virtual network devices. The advantages of Virtualbricks stem from the integration of these components in a flexible and powerful interface. For example, a complete testbed consisting of many interconnected VMs, with their own file systems, can be saved in just one tar file and exported to other PCs or made available for download on a web site.

2 Virtualbricks General Description: “Main” Window and “Bricks”

In this section we will first examine the Virtualbricks main window, to give a general idea of Virtualbricks features; then we will focus on the “bricks”, i.e. the VMs and the VDE elements that are at the core of each project, and on their configuration.

2.1 “Main” Window

The Main window is the operating centre of Virtualbricks. There are five different pages and several command menus (Fig. 1). “Bricks” is the first page and, once selected, it shows an icon for each brick of the current project. A new brick can be added by pressing the “New brick” button and then by selecting the wanted brick from the selection page (Fig. 2).



Icon	Status	Type	Name	Parameters
	running	Switch	sw1b	Ports: 32
	running	Qemu	vm1	command: /usr/bin/kvm, ram: 128, eth0: switchwrapper_port, eth1
	running	Qemu	vm3	command: /usr/bin/kvm, ram: 128, eth0: switchwrapper_port, eth1
	running	Qemu	vm2	command: /usr/bin/kvm, ram: 128, eth0: switchwrapper_port, eth1
	running	Switch	sw2	Ports: 32
	running	Switch	sw4b	Ports: 32
	running	Switch	sw3	Ports: 32
	running	Netemu	vm1tovm2	Configured to connect sw1a to sw1b
	running	SwitchWrapper	switchwrapper	/var/run/switch/sck

Fig. 1. The Home window of Virtualbricks.

Once selected, an existing brick can be powered on/off, configured, deleted and renamed. The second page of the main window is “Events”. It is used to set commands (e.g. a script) that can be executed by Virtualbricks in an autonomous way. The third page, “Running”, lists the running bricks. As in the first page, a number of operations can be performed on the brick selected (opening and closing the configuration terminal, sending ACPI signals, killing processes, etc.). The “Topology” page shows an

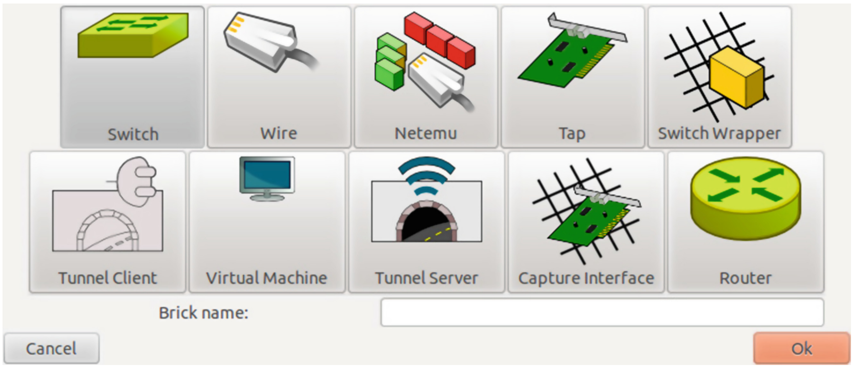


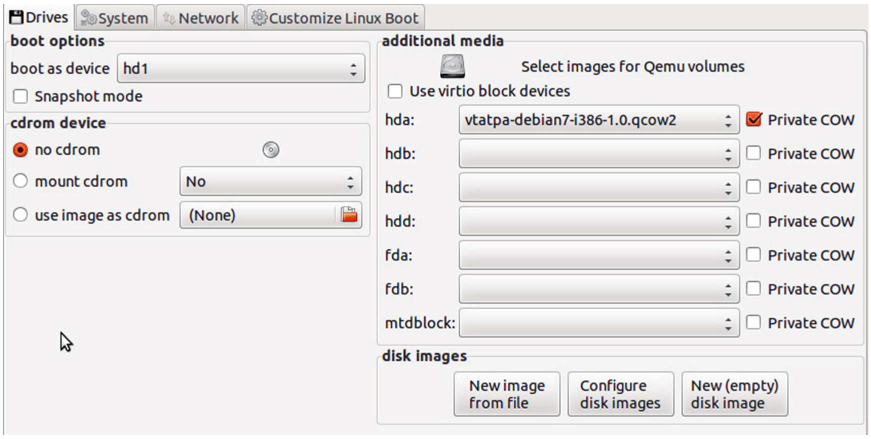
Fig. 2. The Brick selection page of Virtualbricks.

interactive image of the entire topology, which can be useful for visually checking the brick connections. The topology layout can also be exported as an image file. The latest page "Readme", allows either reading or writing comments on the testbed. It can be useful when a testbed is distributed to other researchers or to students.

Virtualbricks supports two kinds of virtual machines: Qemu and KVM. Qemu [13] is a processor emulator and supports a large variety of guest OS, including those built for CPU architectures that differs from that of the host machine. KVM [5] is a virtualization solution for x86 processors based on hardware virtualization technologies (Intel Virtualization Technologies® and AMD-Virtualization®). The full virtualization approach of KVM, when compared to CPU emulators, offers many advantages in terms of guest performance and access to paravirtualized devices, such as virtual disks and virtual Ethernet tools. Moreover, scalability is greatly improved by the joint use of KVM and Kernel Samepage Merging (KSM) [18], a Linux feature that combines multiple identical memory pages that are in use by different processes into a single physical RAM page ("overcommit" feature). On the other hand, KVM needs a CPU with the virtualization extensions and does not support binary translation [19]. Due to their superior performance, KVM is generally preferable whenever there is no need to emulate a different architecture.

2.2 VM Brick Configuration

The VM brick configuration panel can be opened by clicking on the brick with the right mouse button and selecting "Configure". The VM configuration options are grouped into different pages, each representing a different section. The "Drives" page (Fig. 3) enables the configuration of all the block devices, in the form of either virtual disk images or direct access to host peripherals such as CD-ROMs. Disk images cannot be used concurrently with write-on permission by more than one VM, to avoid the corruption of image content. To circumvent this problem, Virtualbricks stores local images in a database. Virtual disk images can be managed directly from Virtualbricks. In particular, a differential disk image can be used by checking the "Private Cow" box.


Drives | System | Network | Customize Linux Boot

boot options

boot as device: hd1

☐ Snapshot mode

cdrom device

[✓ no cdrom]

mount cdrom: No

use image as cdrom: (None)

additional media

Select images for Qemu volumes

☐ Use virtio block devices

hda:	vtatpa-debian7-i386-1.0.qcow2	✓ Private COW
hdb:		☐ Private COW
hdc:		☐ Private COW
hdd:		☐ Private COW
fda:		☐ Private COW
fdb:		☐ Private COW
mtdblock:		☐ Private COW

disk images

Fig. 3. The Drives configuration page of a VM brick.

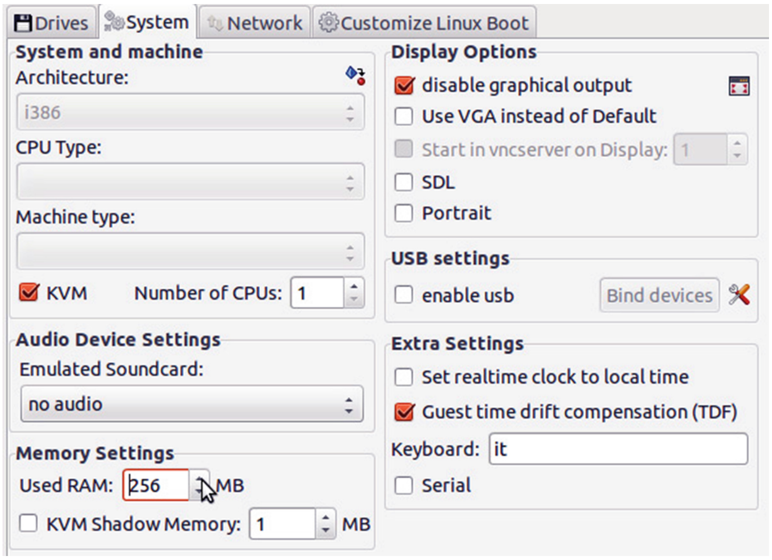
This enables the use of only one single image for all VMs, thus greatly improving the consistency and the dimension of the project. Cow files are used in this case only to save machine dependent configurations. Another important check box is “Snapshot Mode”. If checked, no changes are saved when the VMs are switched off, which can be useful in an educational Lab or when new software is tested.

The “System” page (Fig. 4) contains all the settings related to the VM hardware abstraction. If KVM is checked, KVM instead of QEMU is used and the menu for the selection of the guest machine architecture is disabled, as the only possible architecture is that of the host. The amount of RAM available must be specified in the “RAM” field. In the “Display options” section, it is possible to choose whether and how a virtual monitor should be generated. By default (no check box selected), a new window with a text terminal interface is created. Other options are “Disable graphical output”, or “Start in VNC server”, which creates a new VNC server process on a standard VNC display port, identified by the number specified in the text area. Among the other options in the same page, there is the possibility to synchronize the VM clock with that of the host, a feature that is very useful in dealing with the evaluation of time sensitive protocols, like TCP.

The “Network” page is used to add, remove, configure and interconnect the NIC interfaces. Finally, the “Customize Linux Boot” page is normally used only to debug a Linux kernel running on the guest.

2.3 VDE Bricks

VDE is an Ethernet compliant virtual network designed to interconnect VMs and real computers in the same Local Area Network (LAN) [8, 9]. It consists of many independent tools, such as switches, channel emulators, tap devices, tunnel servers and clients, etc., running on the host. It must be stressed that, like Qemu and KVM, VDE is a software package independent form Virtualbricks. Once VDE is installed on the host, Virtualbricks offers the user a GUI to manage VDE tools, seen as “VDE bricks”, and



System and machine
Architecture: i386
CPU Type:
Machine type:
[✓] KVM   Number of CPUs: 1

Audio Device Settings
Emulated Soundcard: no audio

Memory Settings
Used RAM: 256 MB
[ ] KVM Shadow Memory: 1 MB

Display Options
[✓] disable graphical output
[ ] Use VGA instead of Default
[ ] Start in vncserver on Display: 1
[ ] SDL
[ ] Portrait

USB settings
[ ] enable usb   Bind devices

Extra Settings
[ ] Set realtime clock to local time
[✓] Guest time drift compensation (TDF)
Keyboard: it
[ ] Serial

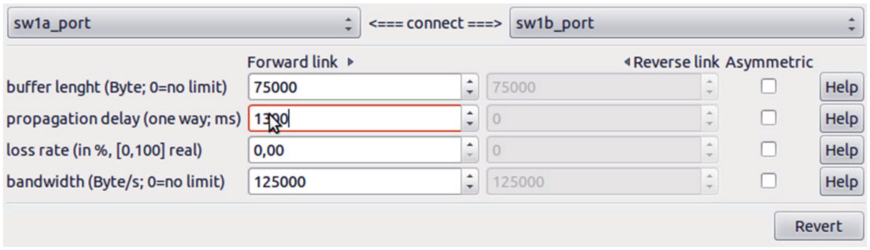
Fig. 4. The System configuration page of a VM brick

thus the possibility to interconnect VMs in many ways. This paves the way to the building of complex testbed layouts, which is what really differentiates Virtualbricks from all other VM management platforms. Of course, as in all virtual testbeds, there will be stringent limits to the maximum bandwidth of VDE tools supported, due to host hardware, CPU load and to the VDE software itself. However, as bandwidths of a few tenths of Mbit/s can be reached, there are plenty of possible applications, except the emulation of high-speed networks.

As in a real Ethernet, the most important tool is the VDE Switch, which emulates a real switch and is the most common way of interconnecting other VMs or VDE bricks. Moreover, by means of a VDE Tap brick, which provides the abstraction of an Ethernet device on the host machine, it is possible to connect the host to the switch, thus enabling the creation of a variety of hybrid testbeds: many VMs plus the host, distributed virtual testbeds, a virtual testbed linked to a real network through its host, etc. Alternatively, it is also possible to start a VDE Switch on the host independently from Virtualbricks and connect a Switchwrapper brick to it. This is the solution preferred by the authors to have remote access to the VMs by SSH (Secure SHell), which is very useful especially if the virtual testbed runs on a remote host. Another essential tool in networking evaluations is the VDE Wirefilter, usually inserted between two switches. Despite its name, it is a channel emulator. Like channel emulators on real machines, e.g. NistNet, Dummynet etc., it aims to reproduce the characteristics of a real wired or radio link, by making it possible to add packet delays, losses, bandwidth limitation, random duplication of packets, MTU restrictions and even the random flipping of single bits. It also supports asymmetrical channels. In Virtualbricks 1.0, however, we use an enhanced version of it, called Netemu, with increased reliability, not included in VDE (it can be downloaded by our web site). This is just a temporary solution in view of the release of next enhanced versions of VDE. Another brick is the VDE Tunnel (client and server), which allows two virtual testbeds, on two different host machines, to communicate on a blowfish encrypted channel, by the VDE tool Cryptcab.

2.4 VDE Brick Configurations

VDE bricks can be configured in two alternative ways: graphical, from a simple configuration interface provided by Virtualbricks (see for example the configuration page of Netemu in Fig. 5), or textual, by entering the VDE configuration commands with the usual syntax from the control console of the brick. In the former case, GUIs can be opened as for VM bricks, by selecting a brick in the "Bricks" page of the home window, pressing the right mouse button and selecting "Configure". The alternative configuration terminal can vice versa be opened by selecting a brick in the "Running" page of the home window, pressing the right mouse button and selecting "Open Control Monitor". Note that once opened, it must be closed by clicking on the close window check box (be careful because entering the "exit" command does not close the window but switches off the brick). This textual configuration mode is preserved in Virtualbricks to allow advanced users already familiar with VDE commands to continue to use VDE syntax, which is less intuitive but sometimes more powerful than GUIs.



sw1a_port	<=== connect ===>	sw1b_port		
	Forward link ▸	◂ Reverse link	Asymmetric	
buffer lenght (Byte; 0=no limit)	75000	75000	☐	Help
propagation delay (one way; ms)	1300	0	☐	Help
loss rate (in %, [0,100] real)	0,00	0	☐	Help
bandwidth (Byte/s; 0=no limit)	125000	125000	☐	Help

Fig. 5. The configuration page of Netemu brick.

3 Building a Virtual Testbed: DTN Satellite Communications in Space

One of the most important advantages of Virtualbricks over a real testbed is that it is possible to change the testbed layout simply by opening a different project. From the "File" menu it is possible to open an existing project file, create a new one, change the name of the current project, export the entire project (included VMs file systems) into just one file, or vice versa to import a project from a file. These features result into an extraordinary flexibility and easiness to use, for both research and education purposes. Since in virtual testbeds the available processing power of the host machine is a critical factor, as it limits performance, in Virtualbricks only one project can be opened at a time and only one Virtualbrick instance can be run on the same host.

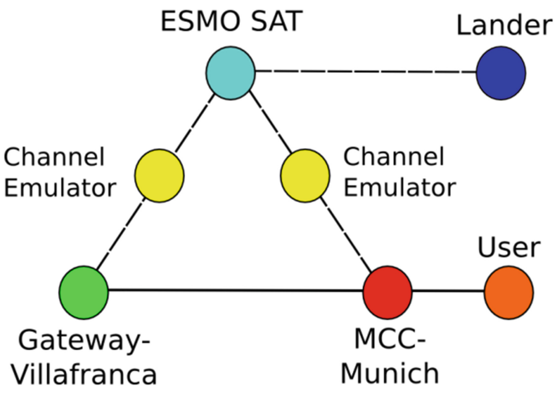


Fig. 6. The logical topology of the ESMO testbed.

Let us now introduce as an example the virtual testbed used in [17], to some extent inspired by the ESMO (European Student Moon Orbiter) ESA mission [20]. Its logical layout is shown in Fig. 6.

3.1 Brick Selection

To build it, we have to open a new project, which will call ESMO. Then we need to select the bricks. As the topology consists of five DTN nodes, we need five VM bricks. The fastest way to do this is to create a new VM, configure it, then clone it four times. Minor adjustments to the VMs, such as NIC connections, can be done after inserting the necessary VDE bricks into the project. As the two satellite links between the ESMO satellite, which orbits around the Moon, and the two ground stations (MCC and Gateway) are characterized by long propagation delays and possible random losses, their emulation requires the use of two intermediate Netemu bricks. All other links are

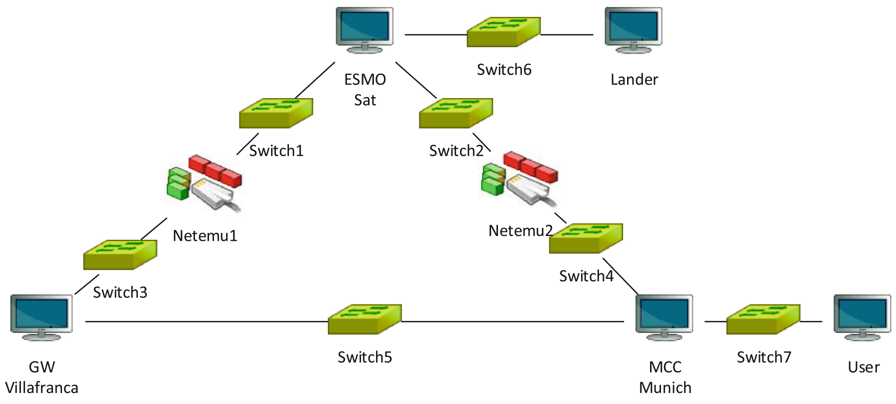


Fig. 7. The Virtualbricks topology of the ESMO testbed.

considered ideal. As direct connections are not possible, VDE Switches are necessary to connect VMs to each other, or a VM to a Netemu brick, thus leading to the Virtualbricks topology shown in Fig. 7.

The five VM bricks will have the same hardware and software configuration, as they are the clones of the same prototype: KVM, with 256 MB RAM, no CD, one hard disk, the same image including the OS and the DTN protocols to be tested, private cows and time alignment with the host (Figs. 3 and 4).

3.2 Brick Connections

In VDE the primary connection element is the VDE Switch, because it is the only brick that has the necessary sockets to connect the plugs of all the other bricks. We can start by connecting the VDE Netemu bricks (and other VDE bricks, if present) to the appropriate switches, by indicating the left and the right switch from the "Plugs" page of the Netemu configuration GUI (Fig. 5). Then, VMs must be connected to the switches. To this end, one NIC must first be created for each connection, through the "Network" page of the VM configuration GUI. Here three parameters must be set: the name of the switch to be connected to, the virtual NIC model and the MAC address. Regarding NICs models, note that the emulation of a real NIC is often partial. For example, the 10/100 Mbit/s NICs do not actually perform any bandwidth reduction, which can be misleading. The MAC address can be inserted either at random or manually. Parameters can of course be changed later, whenever necessary.

3.3 IP Address Assignment

Although in theory the IP address assignment in a virtual testbed is the same as in a real testbed, in practice there are some aspects that deserve to be discussed, to better highlight the scope of Virtualbricks.

In a real testbed the IP address of each machine is usually assigned (at least the first time) from a terminal (i.e. by a user working with a keyboard and a display directly connected to machine). The same could be done in Virtualbricks from VM consoles, once activated from the VM configuration GUI. In practice, when Virtualbricks is operated remotely through a VNC client this solution is impaired by translation problems of characters or difficulty in keeping the mouse control. An apparently simple solution could be to let the user assign the IP address directly from the VM "Network" configuration, as is done for the MAC address. However, it should be remembered that the software configuration of VMs is beyond the scope of Virtualbricks. This is not just a theoretical objection but also a practical one. In fact, all OSs allow the setting of the IP address, but the syntax is different. While the host machine has to run a GNU/Linux (this OS is required to run Virtualbricks), VMs do not.

A possible solution consists in building a control network independent of the experimental links. To this end, in our ESMO testbed we added an additional NIC (eth0) on each guest, with an IP address assigned by a DHCP server on the host. The connection between these NICs and the host is achieved by means of a Switchwrapper brick connected to a VDE Switch running on the host. This control network offers an easy access to all VMs by SSH as soon as they are created, from either the host or whatever node on Internet (provided that the host itself is on Internet, of course). Moreover, this control network is also very useful when experiments are carried out, as it provides the user with a parallel access system independent of the network under study, thus avoiding any possible interference.

4 Virtual vs. Real Testbed: Validation of Virtual Results

As Virtualbricks is a frontend for Qemu/KVM and VDE, the accuracy of its results actually depends on the reliability of these two components. An in-depth assessment of these was previously carried out by the authors in [21, 22] and would be beyond the scope of the present paper. However, for the sake of completeness, let us present a brief comparison between some results presented in [17], obtained with the Virtualbricks testbed described here, and those achievable by a real equivalent. A selection of Virtualbricks results from Fig. 7 of [17] is given in Fig. 8. The aim of the experiment was to assess the ability of ION CGR (Contact Graph Routing) [23], a DTN routing protocol designed by NASA for scheduled intermittent connectivity, to take the right decisions in the presence of parallel paths. Here from ESMO Sat to MCC, the routing alternative is either via the Gateway or directly to MCC. To this end, ten bundles [15] are first generated and taken into custody [14] on the Lander; when the first Lander-Sat contact starts, at 20 s, the first six are transferred to Sat and taken in custody; they are then delivered to MCC via Gateway when the Sat-GW contact opens, at 70 s; the other 4 bundles are transferred to Sat when the second Lander-Sat contact starts, at 100 s; then they are directly delivered to MCC when the Sat-MCC contact begins, at 150 s.

To replicate the test on a real testbed, we have set-up a real equivalent, consisting of 7 GNU/Linux machines (one for each VM, plus two for the channel emulators), running exactly the same code. Results are reported as x-crosses for comparison. The accuracy of the results achieved on the virtual testbed is evident, thus confirming once again that virtualization technologies can be used to carry out experiments on DTN satellite networking, where transmission rates are relatively low and in line with the present limits of virtualized links.

Of course, the higher the Tx rates and the higher the number of VMs, the higher the chances of reaching the limits of virtualization. For this reason, we do not suggest using virtualization technologies for high speed networks; we do however deem virtualization a perfect match for satellite communication in general, because of low Tx rates, and with DTN in particular, because of both low Tx rates and link intermittency. The unavailability of many links for relatively long periods, typical of most DTN environments, including LEO sat communications and deep space networks, results in intermittent use of VMs, i.e. in a lower computational load for the host machine. In other words, the processing power of the host CPU is shared only by the fraction of active VMs, which can be small even in a large testbed.

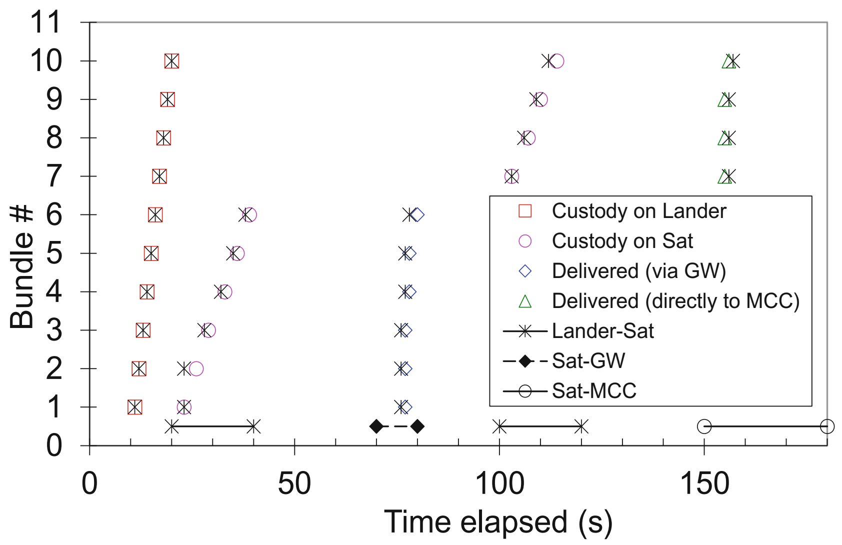


Fig. 8. Bundle transfer from Lander to User. Markers: Virtualbricks; x-crosses: real testbed; segments at the bottom: contact windows.

5 Research and Education with Virtualbricks

The testbed shown in the previous section was successfully used with minimal modifications in further research on Moon communications [24] and in LAB activities of the course [16]. Other Virtualbricks testbeds were used to develop and test DTNperf_3 [25], and more recently to carry out joint research on DTN routing with SPICE center of Democritus University of Trace (Greece). This clearly proved the potential of the tool for both research and education.

For fairness, we also remember the limits of virtualization. They consist in the unfeasibility of carrying out high-speed tests and in software maintenance, which is only apparently the same as in a real testbed. In fact, the possibility to "buy" unlimited VMs at no cost, may lead to the unnecessary proliferation of testbeds, whose software maintenance requires the same effort of their real equivalents (e.g. for updating OS or applications). The informed user, aware of this risk, will easily avoid the problem.

For the reader convenience, let us conclude this section by listing the advantages provided by Virtualbricks in both research and education.

5.1 Research

- Reduced TCO (total cost of ownership): no need to buy dedicated hardware.
- Use of real protocol stacks, by contrast to network simulators.
- Very good performance by using KVM for VMs, by contrast to emulators; alternatively, high flexibility in the CPU architecture choice by using Qemu.

- Perfect results reproducibility: as a Virtualbricks testbed is software defined, two independent research team can actually work on the very same testbed and found the very same results.
- Increased productivity: no more one real testbed to be time-shared among many researchers of the same team, but independent testbeds to be used in parallel.

5.2 Education

- Reduced cost: no need of dedicated LABs.
- Increased teacher productivity: no need to maintain/update/change the configuration of a real testbed; no need to organize testbed remote access and sharing among the students.
- Easy installation: Virtualbricks is in Debian and can be installed with the usual commands (apt-get install Virtualbricks). Note, however, that at present ver.1.0, just released, must be downloaded from [12].
- No need to set-up testbeds: once installed Virtualbricks, it is easy to import pre-configured testbeds provided by the teacher.
- No more one testbed fits all: a set of pre-configured testbeds (e.g. one different testbed for each LAB activity) can be downloaded by students from a course web site and easily imported into Virtualbricks.
- Increased student productivity and freedom; students can focus their attention on the aim of the LAB activity, without being distracted by the many practical problems related to the remote access and time-sharing of a physical testbed; the presence of a testbed in their own PC allows students to replicate LAB activities at home, or to carry out their own experiments, at their will.

6 Conclusions

In the paper the main features of Virtualbricks have been presented. This virtualization solution for Linux differs from others because of the support not only of VMs (Qemu and KVM), but also of VDE tools, which makes Virtualbricks particularly useful for designing and managing testbeds consisting of multiple interconnected VMs. For this reason, in the description of Virtualbricks the virtual testbed used by the authors in recent research on DTN satellite communications has been considered, as a real application example. The comparison of Virtualbricks results with those achievable with an equivalent real testbed, has shown an excellent level of accuracy, thus confirming the suitability of the virtualization approach for both DTN satellite communication research and education.

References

1. VMware. http://www.vmware.com
2. SCVMM. http://www.microsoft.com/en-us/server-cloud/system-center/virtual-machine-manager.aspx
3. Paragon VM. http://www.paragon-software.com/home/vm-professional/
4. SolarWind. http://www.solarwinds.com/
5. KVM. http://www.linux-kvm.org/page/Main_Page
6. Loddo, J., Saiu, L.: Status report: Marionnet - how to implement a virtual network laboratory in six months and be happy. In: Proceedings of the ACM SIGPLAN Workshop on ML, pp. 59–70. ACM Press, New York (2007)
7. Loddo, J., Saiu, L.: Marionnet: a virtual network laboratory and simulation tool. In: SimulationWorks, Marseille, France (2008)
8. Davoli, R.: VDE: virtual distributed ethernet. In: Proceedings of ICST/Create-Net Tridentcom 2005, Trento, Italy, pp. 213–220, May 2005
9. VDE. http://vde.sourceforge.net/
10. UML. http://user-mode-linux.sourceforge.net/
11. Mininet. https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
12. Virtualbricks. https://launchpad.net/virtualbrick
13. Qemu. http://wiki.qemu.org/Main_Page
14. Caini, C., Cruickshank, H., Farrell, S., Marchese, M.: Delay- and disruption-tolerant networking (DTN): an alternative solution for future satellite networking applications. Proc. IEEE **99**(11), 1980–1997 (2011)
15. Cerf, V., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., Weiss, H.: Delay-Tolerant Networking Architecture. Internet RFC 4838, April 2007
16. TLC Master course on Architectures and Protocols for Space Networks. http://www.engineeringarchitecture.unibo.it/en/programmes/course-unit-catalogue/course-unit/2013/386378
17. Caini, C., Fiore, V.: Moon to Earth DTN communications through lunar relay satellites. In: Proceedings of ASMS 2012, Baiona, Spain, pp. 89–95, September 2012
18. KSM. http://www.linux-kvm.org/page/KSM
19. Binary Translation. http://en.wikipedia.org/wiki/Binary_translation
20. ESMO. http://www.esa.int/esaMI/Education/SEML0MPR4CF_0.html
21. Caini, C., Davoli, R., Firrincieli, R., Lacamera, D.: Virtual integrated TCP testbed (VITT). In: Proceedings of ICST/Create-Net Tridentcom 2008, Innsbruck, Austria, pp. 1–6, March 2008
22. Caini, C., Firrincieli, R., Lacamera, D., Livini, M.: Virtualization technologies for DTN testbeds. In: Proceedings of PSATS 2010, Rome, Italy, pp. 272–283, February 2010
23. ION code. http://sourceforge.net/projects/ion-dtn/
24. Apollonio, P., Caini, C., Fiore, V.: From the far side of the Moon: DTN communications via lunar satellites. China Commun. **10**(10), 12–25 (2013)
25. Caini, C., d'Amico, A., Rodolfi, M.: DTNperf_3: a further enhanced tool for delay-/disruption- tolerant networking performance evaluation. In: Proceedings of IEEE Globecom 2013, Atlanta, USA, pp. 3009–3015, December 2013