Virtual Integrated TCP Testbed (VITT)

Carlo Caini
DEIS/ARCES, University of Bologna
Viale Risorgimento, 2
40136 Bologna, Italy
carlo.caini@unibo.it

Renzo Davoli
Department of Computer Science, University of Bologna
Mura Anteo Zamboni, 7
40127 Bologna, Italy
renzo@cs.unibo.it

Rosario Firrincieli
DEIS/ARCES, University of Bologna
Viale Risorgimento, 2
40136 Bologna, Italy
rosario.firrincieli@unibo.it.

Daniele Lacamera
DEIS/ARCES, University of Bologna
Viale Risorgimento, 2
40136 Bologna, Italy
root@danielinux.net.

ABSTRACT
Research on TCP performance relies either on simulation programs, which run on a single machine, or on the use of real testbeds, where different machines represent different network nodes and data exchange is made through physical network interfaces. This paper proposes a different solution, with the aim of taking the best of both the cited alternative options. The idea is to exploit the most advanced virtualization technologies to integrate the different devices of a real testbed in a single GNU/Linux physical machine. The Virtual Integrated TCP Testbed (VITT) presented in this paper is the practical realization of this concept. Several virtual machines, fully configurable from the host system, are connected through an emulated network, implemented by means of the software tools provided by the Virtual Distributed Ethernet (VDE) project. A simple web interface allows the user to configure the network layout, set the TCP parameters, launch the experiments and gather the results. VITT is built on the experience achieved by the authors in the design and use of a real distributed testbed (TATPA), from which VITT derives some software components. TATPA results proved essential to assess the present limits of the virtualization approach, i.e. the accuracy of results vs. network complexity.

Keywords
TCP Performance, Testbed, Virtualization, VDE, KVM.

1. INTRODUCTION
The proceedings are the records of the conference. ACM hopes to give these conference by-products a single, high-quality appearance. To do this, we ask that authors follow some simple guidelines. In essence, we ask you to make your paper look exactly like this document. The easiest way to do this is simply to down-load a template from [2], and replace the content with your own material.

In the whole computer network history, TCP has always been the most used reliable transport protocol. Its success is due to its great original design, dated back to 1981, which is still at the basis of the present versions, although many improvements have been introduced during its long career. Nowadays, the study of the TCP behavior in different network environments has become very important to analyze the impact of heterogeneity in network topologies, such as the presence of wireless or satellite links [1]. Satellite systems, for example, pose some serious and peculiar challenges to TCP, due to the long Round Trip Times (RTT) and the possible presence of random errors [2]. To study the impact of heterogeneous networks on TCP, as well as the efficiency of new TCP variants in coping with their challenges, two main approaches can be adopted. The first relies on simulation, the second on real testbeds. Both solutions present advantages and disadvantages. In short, network simulators, as ns-2 [3], Cnet [4], Opnet [5], Qualnet [6], Insane [7] and NCTUns [8], by relying on a single workstation, are easy to set up and maintain but depend on the accuracy of protocol implementations and/or modeling. On the other hand, testbeds provide a test environment much closer to reality, at the expenses of greater cost and much higher maintenance. In most cases (e.g. [9], [10]) testbed end-points (sources and sinks) are PCs, whereas intermediate nodes can be either PCs (e.g. Linux routers) or custom devices (e.g. switches, routers). TCP testbeds rely on emulators, as NistNet [11], to include or reproduce network challenges (e.g. long delays, limited bandwidth, and random losses).

By building on the concept of emulation, this paper shows that (and how) not only the challenging components, but an entire TCP testbed can be fully emulated and integrated in a single modern PC, thanks to the most advanced technologies developed by research on virtual machines (VMs) and networks. The aim is to match the accuracy of real testbeds with the advantages of simulations: lower cost, higher flexibility and reduced maintenance. Last but not least, by relying on software modules running on standard hardware, the reproducibility of the experiments in a virtualized environment is higher than in real testbeds. This idea led us to create the Virtual Integrated TCP Testbed (VITT), described in this paper. In VITT the operating system of the physical machine hosts a limited number of "guest systems", each one in a separate VM. Thanks to a set of tools provided by the Virtual Distributed Ethernet (VDE) project [12], the underlying network behavior can be reproduced by emulating its components: switches, routers and links are replaced with software modules that can be customized to reproduce as closely as possible the real network environment we want to study. Moreover, in VITT the host system also acts as centralized controller and provides the user with a simple web based interface to configure the virtual network components and to guide the user through the experiments.

The idea of exploiting network virtualization to implement testbeds components has recently appeared in the literature. To provide some examples, we cite [13], where the implementation of a

virtual router based on Xen technology is presented, and [14], which describes a hybrid testbed composed of real mesh nodes and a virtualized environment. Virtualization is further exploited in [15] and [16], where VMs are used to build testbeds devoted to the evaluation of security issues. In [17] the aim is to realize an "inbox" protocol development environment. With respect to these examples, the novel aspect of our approach is that VITT aims at preserving the timing accuracy, as it focuses on the evaluation of protocol performance. This may become a challenge due to the limited processing resources of the real machine. Dealing with time sensitive protocols, it is essential to assess the accuracy and the limits of virtualization by performing numerical comparisons with other approaches. To this end, the results obtained with our virtual testbed are compared with those achieved by TATPA (Testbed on Advanced Transport Protocols and Architecture) [10], a real distributed testbed with which VITT shares the aim and some key elements.

2. THE TATPA TESTBED

Before analyzing VITT components, it is useful to give a brief description of TATPA, with which VITT shares the aim and some key elements. The TATPA testbed (Fig.1) [10] primary objective is to reproduce a heterogeneous network where satellite connections have to compete for network resources with wired connections. Of course, it is also possible to study pure satellite environments, by simply disabling background terrestrial components. Satellite connections are composed of both wired legs and a final satellite link, while TCP background traffic is present only in the entirely wired paths. Satellite and wired connections share the R1 - R2 link, whose bandwidth can be limited in order to study congestion effects. The satellite link can be either real or emulated. The testbed allows the user to assess performance of all the TCP variants available in the Linux kernel, as well as to evaluate the validity of alternative approaches, based on PEPs (Performance Enhancing Proxies) [18] or DTN (Delay/Disruption Tolerant Networking) [19].

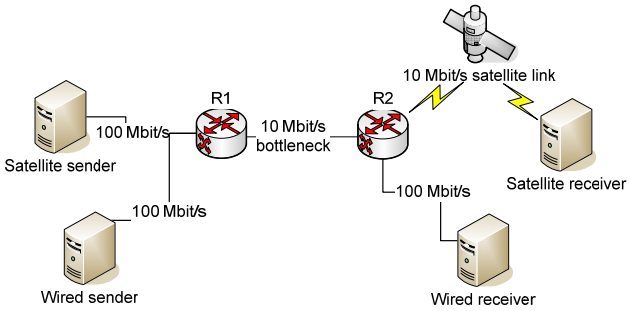

Figure 1: TATPA logical layout.

TATPA consists of about ten Linux PCs (kernel 2.6.20) plus some switches and a switched rack PDU (Power Distribution Unit). In most PCs is installed the MultiTCP kernel patch [20]. A dedicated PC hosts the TATPA controller, which allows the user ubiquitous access to the testbed via a standard web browser. Commands between the controller and the other testbed components are exchanged by means of the RTeCo [10] client/server application.

3. THE VIRTUAL INTEGRATED TCP TESTBED (VITT)

VITT aims to reproduce a real testbed on a single box, giving the abstractions of TCP endpoints and other network components. Endpoints are built as independent VMs: by means of a monitor application (Linux KVM, described in the next sub-section) it is possible to create the abstraction of the hardware components needed to run an operating system. The entire network infrastructure is built using the Virtual Distributed Ethernet [12], a set of applications specifically designed for setting up virtual networks. VDE was designed at the University of Bologna in the context of the Virtual Square (V2) project [21]. The goals of the V2 project include integration, unification and communication among different existing virtualization tools. VDE is the network support of the V2, providing a virtual Ethernet data link layer fully compatible with existing protocols and applications. Like a real network, a VDE is composed of routers, switches, and even wired or wireless emulated physical links. In VITT (Fig.2) the host and VMs are connected to the virtual network through the VDE switch (vde_switch) sockets. Each subnet is connected to the previous and the following one by a single VDE "Layer 3 switch" (vde_l3) a sort of router that simply connects two or more subnets, providing basic static routing and traffic control modules. The links between subnet switches are implemented by means of a configurable VDE wire (wirefilter), which emulates the desired peculiar characteristics of the link. By means of the VDE components, VITT user interface (Fig.3) allows the user to configure the network with all the emulated links, with only a partial constraint on topology, which consists of a first subnet, where the desired number of TCP senders are connected, and of a chain of subnets for receivers. The length of the chain (i.e. the number of subnets) is left to user choice, under the practical constraint that the computing resources of the host machine cannot be exceeded, as discussed later. All the software of the VITT project is distributed under the GPL2 [22] free license. All information about downloading and installation can be found in [21]. The VITT main elements are briefly described below.

3.1 Linux KVM (Kernel Virtual Machine)

There are many software packages implementing VMs, available both under proprietary and free licenses. They can be classified on the basis of the way they provide abstractions of existing or simulated hardware resources to the virtual environment [23]. The approach that makes it possible to run non-native operating systems (i.e. those compiled for another CPU architecture) on the guest machine is called emulation (or full system simulation or even full system virtualization with dynamic recompilation). When the hardware is not simulated, but an API is offered by the host operating system to access real hardware, the approach is called paravirtualization, and the monitor application is called hypervisor (e.g. Xen [24]). Finally, in the native approach (or full virtualization) the VM simulates only the hardware necessary to run a guest OS designed for the same type of CPU. The native approach greatly benefits from CPU virtualization extensions, where available. Intel and AMD have independently developed these extensions to the x86 architecture. They are not directly compatible with each other, but have largely the same function. Either allows a VM monitor to run a guest operating system significantly reducing the performance penalties due to the emulation of the CPU operations. Linux (from 2.6.20 version) supports a VM monitor called Kernel Virtual Machine [25]. KVM too is considered a hypervisor, since it runs on a modified host operating system; it offers full support for x86 CPU virtualization extensions and its developers are currently working on the implementation of paravirtualized I/O devices. KVM plays a key role in the whole VITT project, since it allows the host machine to run many VMs with acceptable performance, provided that the virtualization extensions are implemented on the host machine CPU. A possible methodology for evaluating the guest systems performance and discovering the resource limits of the host machine is proposed later in this paper.

The KVM hypervisor is actually a modified Qemu [25] that can exploit the features offered by the underlying kernel. Qemu is fully supported by the VDE network, via a special command line front-end called vdeqemu. Virtual network cards can be connected to switches by adding an option to this command. Note that a limit on the available bandwidth between VMs may come from the implementation of the virtual network interface on the guests. Though current KVM host-to-guest maximum bandwidth is roughly 50Mbit/s, quite enough for the VITT current application field, future KVM releases including better paravirtualization should push this limit to make it comparable to the speed normally achieved via the loopback device (over 200Mbit/s).

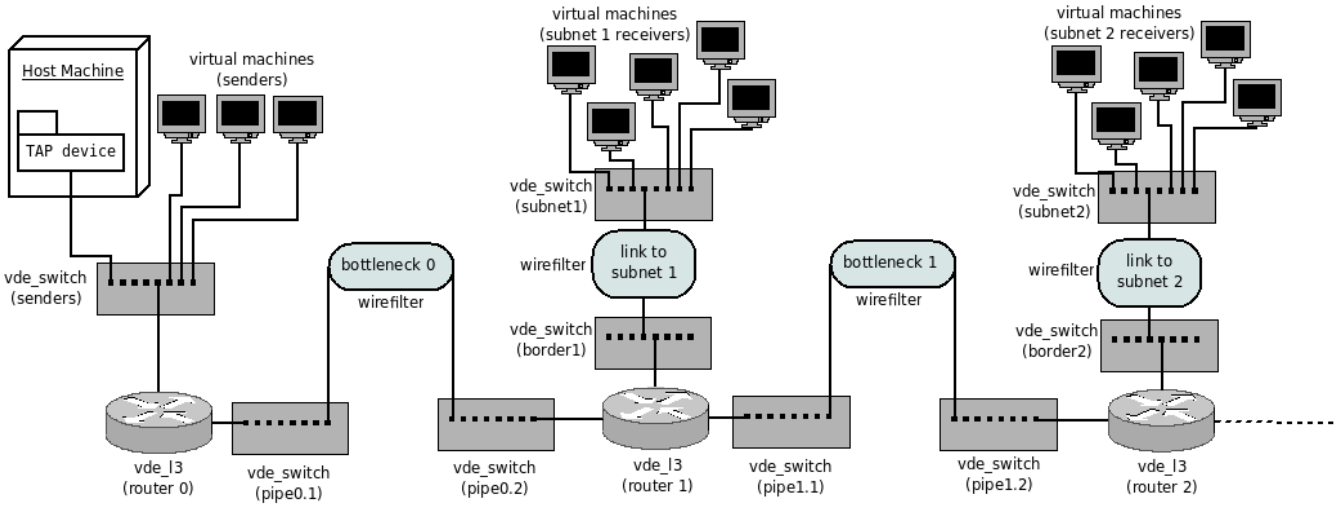

Figure 2: Block diagram of VITT components.

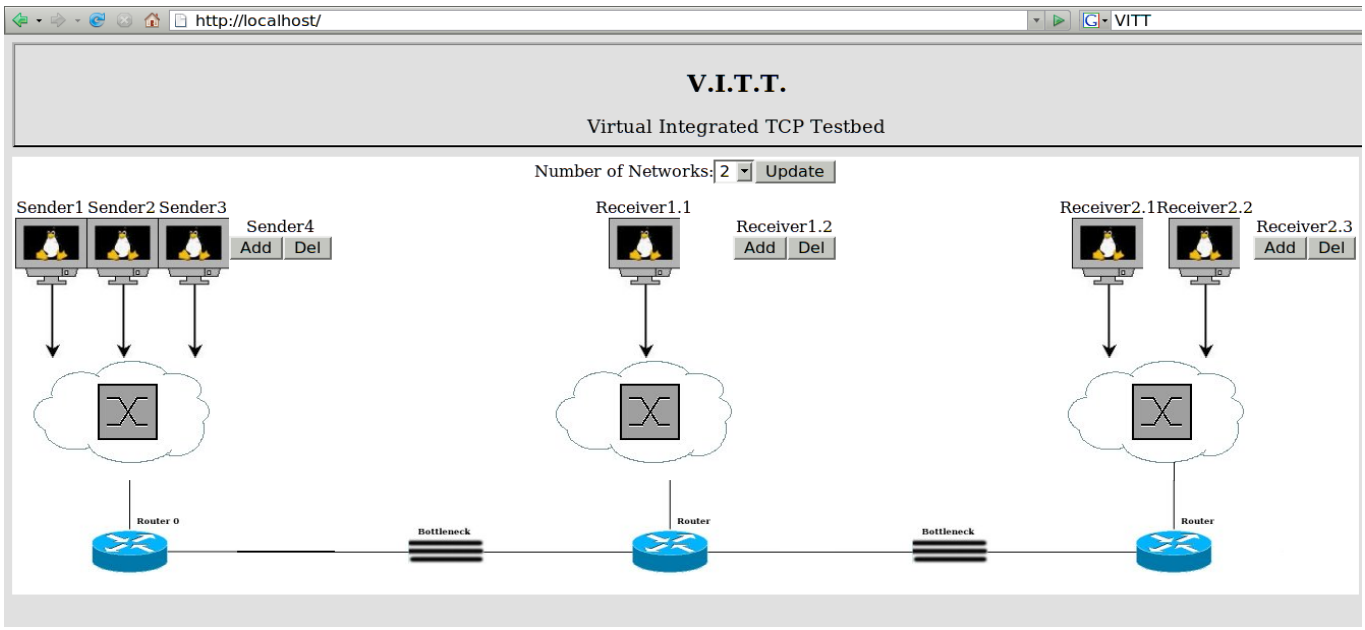

Figure 3: Snapshot of the VITT configuration web page.

3.2 VDE Layer 2 switch and tun/tap device (vde_switch)

When VITT is started, each VM operating system boots up and runs with its own specific network configuration, depending on the testbed layout. Each VM has a virtual network card, which is connected to a vde_switch before the VM is turned on. The vde_switch has a lot of features: it supports separate VLANs and the fast spanning tree protocol [26]. Moreover, the vde_switch provides connection with both virtual and real machines. In the former case, there are many ways to connect VMs to the switch, depending on which manager or hypervisor is adopted. In the latter case, it is necessary to use the tun/tap device on the host system to connect it to the switch. In this way, the real host machine can use TCP/IP to coordinate the test and exchange commands and data with the guests. The host system is connected to the sender switch and all the routes needed to reach the receivers are added on the host machine.

3.3 VDE Layer 3 switch (vde_l3)

Each subnet central switch is connected to a VDE Layer 3 switch, which conforms to [27] and [28]. The neighbor machines set the routes to the other subnets by using their local subnet vde_l3 IP address as a gateway. While it receives traffic from any of its interfaces, vde_l3 forwards every packet, choosing the target address from its internal routing table, and replacing the

destination MAC address with that of the chosen neighbor. Moreover, vde_l3 has a set of plug-ins for different types of traffic control algorithms. This feature has been specifically added for VITT, to test how intermediate node packet queuing disciplines can affect the higher layers. At present, drop-tail packet FIFO, a simple token bucket or the Random Early Detection (RED) [29] can be used to shape outgoing traffic on virtual network devices.

3.4 VDE wire (wirefilter)

Wirefilter works as a special cable interconnecting two vde_switches. It can emulate several real physical network characteristics, such as delays, bandwidth limitations, packet losses, random duplication of packets, MTU restrictions and even flipping of single bits. Virtually, any wired or wireless physical link can be emulated with high accuracy, thanks to its support for asymmetrical channels and configurable random variation of parameters. VITT relies on wirefilters to interconnect consecutive subnets as well as VMs and routers of the same subnet (Fig.2). Wirefilter is used in VITT both to emulate satellite channels and to create congestion on bottlenecks.

3.5 VITT User Interface

VITT can be fully controlled either locally or by remote through a dynamic web user interface, written in PHP and installed in the host machine. We can distinguish three phases in the execution of a test. The first step consists in building and configuring the network layout through the layout configuration page (Fig.3). Current settings are saved for future use. The second step is started by accessing the main page, which, after performing a testbed control, shows a picture of the previously configured network layout. By clicking on the icon of a TCP endpoint, a dynamic page is opened, which allows the user to modify the TCP related sysctl parameters for that VM. The process, performed by invoking remote shell scripts via RTeCo, is transparent to the user. Finally, the test page guides the user step-by-step through the definition of the experiment parameters. Each sender can be selected to initiate one or more TCP file transfers, or, alternatively, to initiate a UDP flow at the desired bit rate, towards one of the receivers. Once the experiment has finished, TCP connection details provided by MultiTCP are saved on the host machine, and a preview of some charts can be automatically displayed using xgraph [30].

4. VIRTUALIZATION APPROACH VALIDATION

The VITT virtualization approach relies on the basic assumption that the hardware resources are not too scarce to support the computational load of the experiments. The validity of this assumption needs to be verified, as well as the accuracy of the results achieved. In the following, the causes of hardware limits will be discussed in depth, before directly comparing VITT and TATPA results.

4.1 Host machine resource limits

Running several VMs on a sole Linux box is quite a demanding process in terms of hardware resources. Even if the host machine is equipped with a cutting-edge technology multiprocessor CPU, with full support of virtualization extensions, many problems may easily emerge. A physical constraint on the number of parallel VMs is posed by the amount of RAM installed on the host. To alleviate this problem, we managed to reduce to only 32 MB the virtual RAM necessary to run each VM. Of course, to this memory must be added the memory necessary to run the host operating system itself and all the VDE virtual network components, which run as applications of the host system. Even more stringent and difficult to estimate are the timing requirements. In fact, an integrated virtual networking testbed, like VITT, must emulate all the testbed components relying only on the processing power of a sole machine. Processing overloads, or in general shared resources among virtual entities that are unrelated and independent in the real world, can disturb the timing evolution of communications or distributed computations in the emulated environment. It is worth noting that the importance of these possible timing problems is largely dependent on the experiment target. For example, virtualization testbeds evaluating distributed applications running on Internet are usually unaffected by timing issues. In this case, the need for processing and communication facilities can be a problem only when virtualizing large sets of VMs or interacting processes. By contrast, dealing with performance evaluation of time-sensitive protocols, as in VITT, even relatively small synchronization losses among clocks of the host and VMs can influence the protocol behavior and the performance evaluation in a relevant way. In particular, as long as some experimental TCP features (e.g. packet pacing) require a timer granularity of one millisecond on the endpoints, the host operating system must be able to switch between VM processes quickly enough, in each scheduler epoch, in order not to miss any software interrupt in the guest machines, which could cause timer drift effects. To assess the resource limits of the specific host machine on which VITT is installed, VITT has been provided with a specific benchmark tool, to be run before executing tests. This benchmark starts by setting up a single VM, putting it and the host machine under stress by performing a tight loop of computations and checking clock alignment. Then the test is repeated by adding a second VM and so on, until any of the guest operating system timers shows a significant drift. Of course, the faster the host machine, the larger the number of VMs needed to cause timer drift effects serious enough to compromise experiment validity. This benchmark is useful in finding an approximate ceiling on the number of VMs supported by the host machine hardware. As an example, Fig.4 shows the results of benchmark execution on the hardware platform used to develop and validate VITT.

In the chart, the average and the maximum timer drift registered on the VM set is plot versus its cardinality. Data are normalized to single test length (120 s). It is interesting to note that, although the time drifts for the first five VMs are marginal (only a few milliseconds), inserting a sixth guest into the system leads to a clearly unacceptable timer inaccuracy, in one (or more) of the guests, revealing that the Linux process scheduler is no longer able to tackle all the guest processes without introducing idle times.

4.2 Validation of VITT results

The reliability of VITT results basically depends on three factors: the complexity of the emulated network, the power of available hardware resources and the accuracy of the software virtualization tools (i.e. VMs and VDE elements). Regarding the

network, our primary aim is obviously to assess VITT accuracy in the same satellite environment already considered in TATPA. To

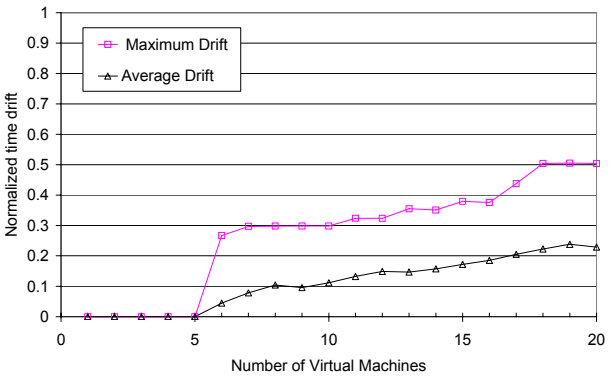

Figure 4: Time drifts normalized to the lenght of the experiments (120 s) vs the number of VMs.

this end, VITT was configured to reproduce the butterfly layout of TATPA given in Fig.1. Note that this task can be accomplished by using only four VMs, necessary to emulate end-points, while R1 and R2 routers, as well as the satellite link, can be emulated by VDE elements (L3 switches and wirefilters), which run on the host machine. Four VMs is a figure compatible with the indication given by our evaluation tool on the hardware platform actually used in our tests. This is at present at the cutting edge of hardware virtualization technology, mounting an "Intel® Core™2 Quad" and 4 GB of RAM. As far as virtualization software is concerned, in the comparative tests presented here the latest KVM and VDE implementations available (KVM release 48, VDE unstable development version based upon 2.1.6) were adopted. Note that the following results should be considered preliminary, as their accuracy tends to improve as new versions of the virtualization software are released.

A preliminary comparison of VITT and TATPA results was carried out by considering two different environments. The first series of tests (Fig.5) refers to a "pure satellite network", where the performance of a single TCP satellite connection is limited only by the presence of random errors on the satellite channel (PER=1%, 10 Mbit/s).

The data (averaged over multiple tests) emphasize the different RTT impacts on performance for different TCP variants. TATPA data are shown as lines, while VITT results are represented as markers. The agreement is clear for all the three TCP variants examined, namely, SACK [31], BIC [32], and Hybla [33].

The second series of tests (Fig.6) refers to a "heterogeneous network", where a satellite connection (variable RTT) has to compete for bandwidth resources on the R1-R2 bottleneck (Fig.1) with terrestrial connections (RTT=25s, PER=0%). The aim of this test is to show the different responses given by different TCP variants to the RTT unfairness problem. Although the level of agreement is lower than before, it is still more than satisfactory. Also in this case, random effects are averaged on multiple tests.

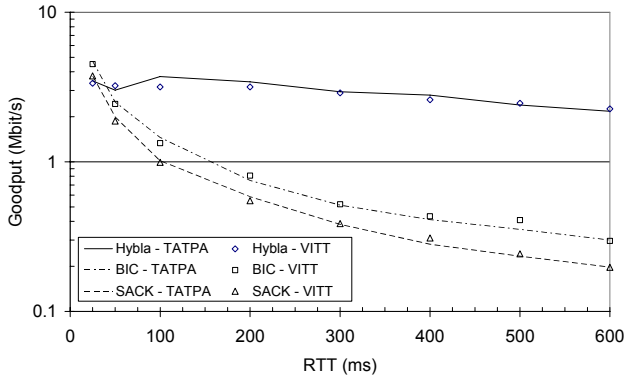

Figure 5: Comparison of TATPA and VITT numerical results. Performance of a single satellite connection in presence of a residual packet error rate (PER=1%); no congestion.

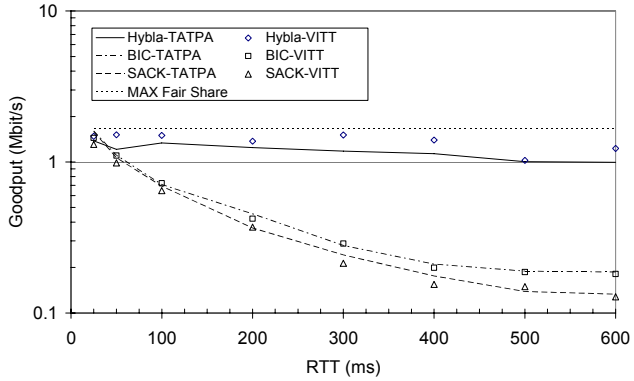

Figure 6: Comparison of TATPA and VITT numerical results. Average performance of 3 satellite connections, in presence of a residual packet error rate (PER=1%) and congestion on the R1-R2 bottleneck (3 SACK wired connections, RTT=25 ms).

Finally let us conclude the comparison between TATPA and VITT with some remarks on costs. Consisting of standard "bookshelf" components, the VITT hardware platform used in the validation tests costs marginally more than a standard desktop. As a result, considering that in VITT the controller is integrated in the host machine, while in TATPA it is external, the total set-up cost of VITT hardware is cut about an eighth with respect to TATPA. The total cost of ownership is likely to fall even more, as VITT maintenance is almost nothing, while it is very time-consuming in TATPA. In general, the savings in cost and maintenance are obviously dependent on the number of VMs supported by the hardware platform. These are bound to increase with the amount of parallelism introduced in new CPU architectures (e.g. 8 cores CPUs).

5. CONCLUSIONS

In this paper the virtualization concept has been applied to implement an integrated virtual testbed, VITT, devoted to performance evaluation of TCP variants on heterogeneous networks including satellite legs. The aim of this innovative approach, made possible by VDE software tools, is to match the flexibility and accuracy of real testbeds with the limited cost and maintenance simplicity of network simulators. Preliminary results are encouraging, as they show a good agreement between VITT

results and those achieved by a real distributed testbed. Remarks on the limits of virtualization vs. network complexity and hardware constraints complete the VITT performance analysis. Future works will be focused in expanding the total number of VMs available in the testbed. This will be performed both by increasing the number of VMs supported by a single processor (thanks to software and hardware virtualization technology advances), and by physically interconnecting more hosting machines exploiting a distributed approach.

6. ACKNOWLEDGMENT

This work was supported in part by the IST-027393 SatNEx Network of Excellence.

7. REFERENCES

[1] C. Barakat, E. Altman, and W. Dabbous, "On TCP performance in a heterogeneous network: a survey", *IEEE Commun. Mag*., vol. 38, issue 1, pp. 40-46, Jan. 2000.

[2] Y. Hu and V.O.H. Li, "Satellite-based internet: a tutorial", *IEEE Commun. Mag*., pp. 154-62, March. 2001.

[3] Network Simulator ns-2 University of California, Berkeley, available at http://www.isi.edu/nsnam/ns/.

[4] Cnet network simulator: http://www.csse.uwa.edu.au/cnet / .

[5] Opnet network simulator: http://www.opnet.com/.

[6] Qualnet network simulator: http://www.scalable-networks.com .

[7] B.A. Mah, "Insane Users Manual", the Tenet Group Computer Science Division, Univ. California, Berkeley, 1996.

[8] NCTUns: http://nsl10.csie.nctu.edu.tw/

[9] EMULAB: http://www.emulab.net/

[10] C.Caini, R.Firrincieli, D.Lacamera, S.Tamagnini, D.Tiraferri, "The TATPA. testbed", *in Proc. of IEEE/Create-Net Tridentcom 2007*, Orlando, USA.

[11] NIST Net: http://www-x.antd.nist.gov/nistnet/

[12] R. Davoli, "VDE: Virtual Distributed Ethernet", *in Proc. of IEEE/Create-Net Tridentcom 2005*, Trento, Italy, May 2005, pp.213-220.

[13] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, L. Mathy, T. Schooley, "Evaluating Xen for Router Virtualization", *in Proc. of IEEE ICCCN 2007*, Honolulu, Hawaii USA, Aug. 2007, pp. 1256 – 1261.

[14] A. Zimmermann, M. Gunes, M. Wenig, U. Meis, J. Ritzerfeld, "How to Study Wireless Mesh Networks: A hybrid Testbed Approach", *in Proc. of AINA '07*, Niagara Falls, Canada, May 2007, pp. 853-860.

[15] A. Volynkin and V.Skormin "Large-scale Reconfigurable Virtual Testbed for Information Security Experiments", *in Proc. of IEEE/Create-NetTridentcom 2007*, Orlando, USA

[16] D. Duchamp and G. DeAngelis, "A Hypervisor Based Security Testbed", *in Proc. of DETER 2007*, Aug.2007, Boston, USA

[17] X.W. Huang, R. Sharma, and S. Keshav: "The ENTRAPID Protocol Development Environment, *in Proc. of IEEE INFOCOMM'99*, 1999, pp 1107-1115.

[18] C. Caini, R. Firrincieli, D. Lacamera, "PEPsal: a Performance Enhancing Proxy for TCP satellite connections", *IEEE Aerospace and Electronic Systems Magazine*, Vol.22, Issue 8, pp. b7-b16, August 2007.

[19] V. Cerf , A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, H. Weiss "Delay-Tolerant Networking Architecture", *Request for Comment RFC 4838*, IETF, Apr. 2007.

[20] C. Caini, R. Firrincieli, D. Lacamera, "A Linux Based Multi TCP Implementation for Experimental Evaluation of TCP Enhancements", *in Proc. of SPECTS 2005*, 2005, Philadelphia, USA, pp. 875-883.

[21] wiki V2: http://wiki.virtualsquare.org/ .

[22] GNU General Public License v.2. June 1991: http://www.gnu.org/licenses/old-licenses/gpl-2.0.html .

[23] Virtual Machine on Wikipedia: http://en.wikipedia.org/wiki/Virtual_machine .

[24] The Xen hypervisor: http://www.xensource.com/ .

[25] The Linux Kernel Virtual Machine: http://kvm.qumranet.com/kvmwiki Qemu: http://fabrice.bellard.free.fr/qemu/

[26] IEEE 802.1w, "Rapid Reconfiguration of Spanning Tree": http://www.ieee802.org/1/pages/802.1w.html .

[27] D. C. Plummer, "An Ethernet Address Resolution Protocol", *Request for Comment RFC 826*, IETF, November 1982.

[28] F. Baker, "Requirements for IP Version 4 Routers", *Request for Comment RFC 1812*, IETF, June 1995.

[29] S. Floyd, 'Connections with multiple congested gateways in packet-switched networks, part I: one-way traffic,' *ACM Computer Communications Review*, 21(5): 30-47, October 1991.

[30] Xgraph: http://www.xgraph.org/ .

[31] M. Mathis and J. Mahdavi, "TCP Selective Acknowledgment Options", *Request for Comment 2018*, IETF, Oct. 1996.

[32] Lisong Xu, Khaled Harfoush, and Injong Rhee, "Binary Increase Congestion Control for Fast Long Distance Networks", *in Proc IEEE INFOCOM '04*, Hong Kong, March 2004, vol 4, 7-11 pp. 2514 – 2524.

[33] C. Caini, R. Firrincieli, "TCP Hybla: a TCP Enhancement for Heterogeneous Networks", *Int. J. Satell. Commun. Network*, vol. 22, pp.547-566, Sep.-Oct. 2004