# Network Emulator: a Network Virtualization Testbed for Overlay Experimentations

Vincent Autefage, Damien Magoni

# *N*etwork *Emu*lator: a Network Virtualization Testbed for Overlay Experimentations

Vincent Autefage
University of Bordeaux - LaBRI
autefage@labri.fr

Damien Magoni
University of Bordeaux - LaBRI
magoni@labri.fr

***Abstract*—Experimentation is typically the last step before launching a network application in the wild. However, it is often difficult to gather enough hardware resources for experimenting with a reasonably sized overlay application inside a controlled environment. Virtualization is thus a handy technique for creating such an experimentation testbed. We propose a tool called *NEmu* designed to create virtual dynamic networks for testing and evaluating prototypes of overlay applications with a complete control over the network topology and link bandwidths. *NEmu* builds host-based overlay networks by using emulators such as QEMU. We illustrate the use of *NEmu* in the context of a file distribution overlay application. We evaluate the impact of chained TCP connections on the application performances. We show that *NEmu* enables us to obtain performance results concerning data rates and delays for end hosts depending on the number of intermediate hosts and the networking parameters of the overlay.**

***Index Terms*—network virtualization, QEMU, virtual testbed, chain of TCP connections.**

## I. Introduction

Experimentation is important to realistically and accurately test and evaluate a network application. It can be a difficult task when trying to experiment with an overlay application involving dozens of machines or more. Using the Internet as a testbed is impractical as no parameters can be controlled. Setting up a hardware testbed is expensive and cumbersome. Furthermore, overlay applications can have very different ways of connecting hosts to each others and changing the network topology and network parameters of a hardware testbed is time consuming and error prone. Virtualization techniques for creating such an experimentation testbed can save resources and ease manipulations. It is a proved method for reducing the equipment and space costs as well as the energy consumption of using physical hosts [1]. Our solution to overcome the above hardware constraints is thus to build a testbed able to set up virtualized networks. A virtual network uses virtual machines instead of physical hosts and connects them with virtual links in order to build a virtual network topology. The virtual machines of a virtual network can be hosted on one or several physical hosts depending on the number of virtual machines needed and the resources capacities of the physical ones.

We propose a tool designed to create virtual networks for testing and evaluating prototypes of overlay applications with a complete control over the network topology and link properties (bandwidth, delay, bit error rate, etc.). The goal of our tool is to enable the creation of reasonably sized virtual networks while minimizing the number of necessary physical hosts and network equipments needed. It can build host-based overlay networks by using emulators such as QEMU [2]. We have called our tool *NEmu* which stands for *n*etwork *emu*lator because it is able to create an emulated network. It is also a tribute to the name of the QEMU software which is a powerful machine emulator heavily used by *NEmu*. The contributions of our work are as follows:

- A detailed description of our *NEmu* tool which is able to manage a distributed set of virtual nodes and links for emulating any arbitrary network topology (Section II).
- A use case of *NEmu* illustrating the performance evaluation of a file distribution application using an overlay distribution tree made of TCP connections. We provide measurements concerning the achievable data rates and delays depending on the number of intermediate hosts (Section III).
- A state of the art on related and previous work targeted at networking emulation and a comparison of the features provided by *NEmu* with the ones offered by similar alternative virtual networking testbeds (Section IV).

## II. Description of *NEmu*

### A. Overall design

*NEmu* is a 5000-line python program which allows to build a dynamic and distributed virtual infrastructure network. It is based on the concept of the *Network Virtualization Environment* (NVE) [3]. The main characteristic of a NVE is that it hosts multiple *Virtual Networks* (VN) that are firstly not aware of one another, and that are secondly completely independent of each other. A VN is a set of *virtual nodes* connected by *virtual links* in order to form a virtual topology. *NEmu* provides the possibility of creating several virtual network topologies with the central property that a VN is strictly disjoint from another in order to ensure the integrity of each VN.

Thus, *NEmu* integrates characteristics that are fundamental to a NVE: First, the *flexibility and heterogeneity* allows the user to construct a customized topology, with custom virtual nodes and virtual links. The *scalability* allows different virtual nodes to be hosted by different physical hosts in order to avoid limitations of a unique physical machine. The *isolation* decouples the different virtual networks which

run on the same infrastructure. The *stability* ensures that faults in a virtual network would not affect another one. The *manageability* ensures that the virtual network and the physical infrastructure are completely independent. The *legacy support* ensures that the NVE can emulate former devices and architectures. Finally, the *programmability* provides some optional network services to simplify the use of the virtual network (such as DHCP, DNS, etc.).

In addition, *NEmu* includes three important extra properties:

- The *accessibility* which means that *NEmu* can be fully executed without any administrative rights on the physical infrastructure. Indeed, the major part of public infrastructures, like universities and laboratories, does not provide administrative access to their users in order to ensure the security and the integrity of the whole domain. Therefore, the user execution would allow most people to use *NEmu* freely.
- The *dynamicity* of the topology enables node hot-connections which means that a virtual node can join or leave the topology dynamically without perturbing the overall virtual network.
- The *community aspect* of the virtual network provides the possibility for several people to supply virtual sub-networks in order to build a community network like the Internet is.

### B. Network elements

*NEmu* is a distributed virtual network environment which allows users to create arbitrary and dynamic topologies. To this end *NEmu* is based on different building blocks. *NEmu* uses *virtual nodes* connected by *virtual links* in order to create a virtual network topology. A virtual topology can be hosted by one or several physical hosts. The part of the virtual topology laying on a given physical host represents a *NEmu session* which is configured by the *NEmu manager*.

*1) Virtual node:* A *virtual node* for *NEmu* is an emulated machine that requires a hard disk *image* to work. This image is typically provided as a regular file on the physical host machine. Two types of *virtual nodes* currently exist in *NEmu*:

- A `VHost` is a virtual *host* machine (i.e., end-user terminal) on which the hardware properties and the operating system can be fully configured by the user.
- A `VRouter` is a virtual *router* directly configured by *NEmu* and provides ready-to-use network services.

Each virtual node uses a *virtual storage* which can be either a real media (cdrom, hard drive, etc.), a *raw* file or a host directory:

- By making a *Sparse* file which only stores the differences with its original file,
- By making a *Squash* file system which is a read-only raw image,
- By using a *FAT16* emulated interface which directly accesses to a host directory,
- By using a *Virtio* interface [4] which also directly accesses to a host directory.
- By using a *Network Block Device* which enables a virtual node to remotely access to a block device through the real IP network [5].

A `VHost` needs a disk image which must be supplied by the user. This image can be prepared prior to creating the virtual network. Furthermore, one image can be used by many `VHosts` by using *sparse files*.

A `VRouter` is directly configured by *NEmu* and provides several services to simplify the virtual network management: DHCP, DNS, NFS, HTTP, SSH, NTP, Netfilter, dynamic routing protocols (RIP and OSPF), and QoS management with *Traffic Control* [6]. Moreover, it is easily possible to add some new services through a plugin system available in *NEmu*. A router is running a customized image version of *Microcore* which is a lightweight and highly configurable Linux distribution [7]. Such a system typically requires $\sim$25 MBytes on disk and $\sim$100 MBytes in memory with all services running. Services provided by a `VRouter` are optional and can be enabled or disabled before or during runtime.

*2) Virtual link:* A *virtual link* for *NEmu* is an emulated network connection between *virtual nodes*. This emulated connection can either be performed inside the machine emulator of a node (the link thus being attached to this node) or be performed by a dedicated emulation program (not running any system image in this case). Three types of *virtual links* currently exist in *NEmu*:

- A `VLink` is a virtual point-to-point *link* interconnecting two nodes.
- A `VHub` is a virtual multi-point *hub* emulating a physical Ethernet hub and interconnecting several nodes.
- A `VSwitch` is a virtual multi-point *switch* emulating a physical Ethernet switch and interconnecting several nodes.

*Virtual links* typically carry Ethernet frames from one virtual Network Interface Card (NIC) to one or more other virtual NICs. This Ethernet traffic is tunneled between virtual nodes by using TCP connections. Each one of those TCP connections is called a `VLink`. *NEmu* can instantiate an emulator with an empty virtual node (i.e., without any operating system, storage, etc.) which only plays the role of a hub (`VHub`). Such an empty QEMU instance is lightweight with a RAM cost of 7.2 MBytes. Alternatively, *NEmu* can use our *vswitch* program to emulate a network component. The *vswitch* is a 2500-line C++ program that can emulate a `VLink`, a `VHub` or a `VSwitch` (defined as modes). The advantages of using a *vswitch* is that the user can set the bandwidth, delay and bit error rate on any interface in any mode whereas QEMU offers no control over its hub emulation. In addition, a `Slirp` is a special type of link whose purpose is to provide an Internet access to the virtual node. It is an emulation of a NATed access to the real Internet by using the physical host NIC.

Figure 1 shows an example of a *NEmu* managed virtual network. On the left side, two `VHosts` are connected to a `VRouter` through a `VSwitch` by using TCP tunnels. On the right side, two `VHosts` are connected to the above `VRouter`

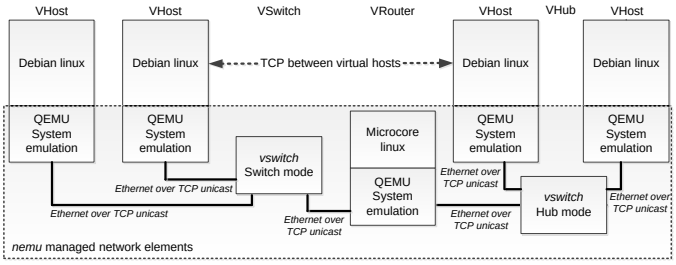


Figure 1. Network elements in action

through a `VHub` by using TCP tunnels. Here, *virtual links* are created and managed inside *vswitch* processes.

### C. Management of virtual networks

As already said above, a *NEmu session* represents a complete configuration of a network topology which lays on a physical host (storages, virtual nodes configurations and links). A distributed virtual network on $n$ physical hosts consists in $n$ *NEmu sessions* at least. A *session* is represented by an auto-generated directory in order to be saved and re-used. A *session* can be saved as a *sparse archive* which compresses all elements and which is compatible with *sparse files* unlike traditional archives.

The *NEmu manager* is the command line user interface to manipulate a *session*. *Sessions* are independent even if they are part of the same network topology. The *manager* can be used in three ways :

- As a python module to be integrated in another script or program.
- As a dynamic python interpreter.
- As a python script launcher.

The *NEmu* manager provides a remote accesses, through SSH connections, to manipulate *NEmu sessions* laying on other distant hosts. The python language is upgraded in order to interact with other distant *sessions*.

## III. Use Case

In order to illustrate how *NEmu* can be used to carry out a performance evaluation of an overlay network application, we consider the case of a file distribution application. Such application typically needs to set up an overlay network among the participating hosts. An overlay network is created when hosts maintain open connections between themselves and streams of data flow along many intermediate hosts before reaching the destination host using some form of flooding or routing mechanisms. Whereas in regular P2P applications, such as file sharing, hosts can be loosely connected, in overlay applications, hosts must maintain a virtual network topology called an overlay topology.

### A. File distribution application

The goal of this application is to send a big file as a continuing stream through a tree-shaped overlay and to deliver the content to several clients placed on the tree's nodes and leaves. Packets are sent only once by the source of the tree. The duplication of packets is done by intermediate nodes implementing multicast on the application layer with connections between nodes set by TCP connections. TCP connections are used to ensure correct data receptions compared to real multicast connections which are on top of UDP.

ROMA [8] also performs multicast distribution for large file in a tree-shaped overlay. This technique is based on managing the size of buffers, at the application layer, in intermediate nodes in order to control data congestion. Unfortunately, this solution increases drastically the global processing delay because as it can drop packets in case of buffer saturation. MCC [9] introduces a similar congestion control mechanism of TCP but at the application layer. This technique is more suitable for unsafe protocols such as UDP.

Our solution consists in only using the back pressure (or windowing) mechanism of TCP to reduce the global tree delivery rate in case of data congestion.

### B. Pipelining TCP connections

The goal of this use case is to obtain preliminary results on overlay connections built by pipelining TCP connections. Indeed, if the stream consists in a video, the data rate is very important to ensure the smoothness of the playback. In the case of a video-conference, the delay is primordial to communicate fluently. To do so, we conduct two different scenarios:

- We firstly want to check that a chain of TCP connections, even with an important number of intermediate nodes, will not degrade the receiving stream of a client. Therefore, we create a branch of the tree in *NEmu* and measure data rate and delay on the receiver interface which is located at the leaf of the branch. We perform the test on only one edge to the extent that we only want to check network performance impacts of a chain of TCP connections.
- Then, we perform another experimentation in where we emulate the complete tree-shaped overlay in order to include the tree outdegree parameter.

A small overlay software which only forwards packets, at the application layer, runs on each node of the experimentation as illustrated by Figure 2.

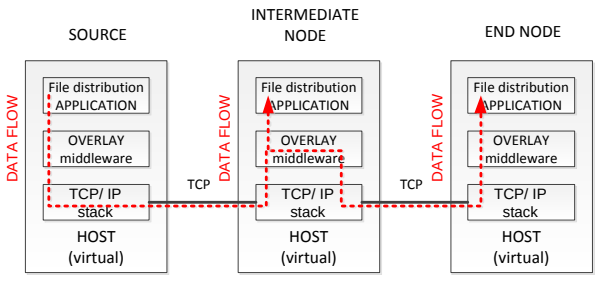


Figure 2. Forwarding mechanism of the overlay by joining two TCP connections

### C. Experimentation settings

We have configured each virtual node as follows: Intel Core 2 Duo Processor, 256 MBytes of RAM, 1 or 2 Realtek rtl8139 NICs, Debian Squeeze 32bits. The simulation was carried out on a single server with 48 CPU cores and 64 GBytes of RAM. Simulation was done several times with different virtual connection bandwidths: respectively 1.25, 2.5, 5, 10 and 20 Mbits/s to verify if the quality of the connection between links have an impact on delay or data rate. These bandwidths were adjusted by using our *vswitch* program.

### D. Chain results

For the chain, we carried out simulations for 2, 4, 8 and 16 nodes in the chain. According to [10]–[12], the average size of the packets is about 500 Bytes but the recent increase in the MTU has led to double the average packet size. Thus, we chose to perform the experimentations with packets of 500 and 1024 Bytes of data.

*a) Data rates:* Results for data rates are presented in Figure 3 and Figure 4. The x-axis represents the number of nodes in the chain and the y-axis represents the client reception data rate in Kbits/sec.

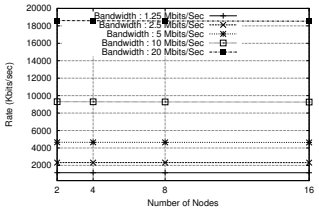


Figure 3. Receiver data rates with packets of 500 Bytes

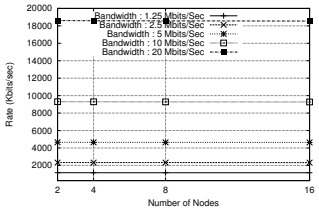


Figure 4. Receiver data rates with packets of 1024 Bytes

We can see that the data rate is not degraded by increasing the number of intermediate nodes. Also, we find that the size of packets has not any impact on the data rate. The small loss compared to the set bandwidth is due to the *vswitch* parameters approximation. Those results ensure that a stream rate will be maintained even if the chain is composed of an important number of intermediate nodes. Consequently, we can assert that $rate_{client} \simeq bandwith$.

*b) Delays:* Results for delays are presented in Figure 5 and Figure 6. The x-axis represents the number of nodes in the chain and the y-axis represents the client reception delay in ms.

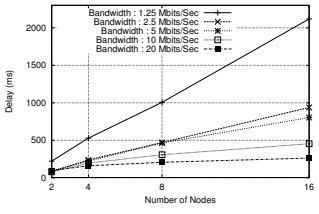


Figure 5. Receiver delays with packets of 500 Bytes

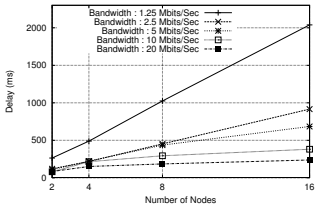


Figure 6. Receiver delays with packets of 1024 Bytes

We can see that the delay increases when the size of the chain grows up. Also, we can see that the phenomenon is more important with a small bandwidth. The reason is that a small bandwidth is more likely to queue some packets due to a lack of available bandwidth. Moreover, the size of packets seems to have negligible impacts on results.

With the help of *NEmu*, we have observed that chaining TCP connections maintains data rate but has an impact on delays. Consequently, chaining TCP connections is not suitable for a real time application especially with a small bandwidth.

### E. Tree results

For the complete tree-shaped overlay, we have carried out the simulations for several depths (1, 2, 4 and 6) and outdegrees (2 to 7) in the tree. We only performed the experimentations with packets of 1024 Bytes due to the fact that this parameter does not have any impacts on results as explained in Section III-D. Results represent the average of data rate and delay between all the clients (i.e. the tree leaves).

*c) Data rates:* Results for data rates are presented in Figure 7 and Figure 8. The x-axis respectively represents the depth and the outdegree of the tree and the y-axis represents the client reception data rate in Kbits/sec.

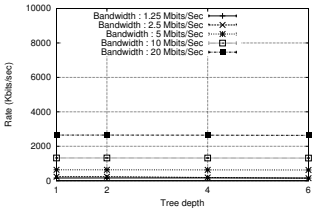


Figure 7. Receiver data rates *vs* tree depth with an outdegree of 7

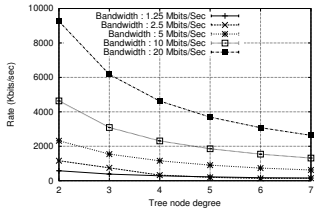


Figure 8. Receiver data rates *vs* tree outdegree with a depth of 6

First, we can see that the data rate is not degraded by increasing the number of intermediate nodes even with a high arity in the tree. Secondly, we can see that the data rate decreases logarithmically when the tree outdegree increases. We can assert that $rate_{client} \simeq \frac{bandwith}{tree\ outdegree}$.

*d) Delays:* Results for delays are presented in Figure 9 and Figure 10. The x-axis respectively represents the depth and the outdegree of the tree and the y-axis represents the client reception delay in ms.

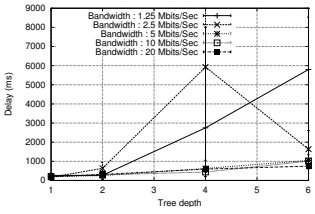


Figure 9. Receiver delays *vs* tree depth with an outdegree of 7

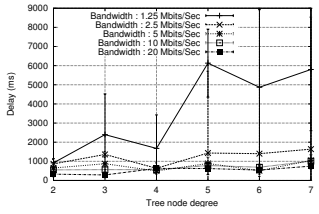


Figure 10. Receiver delays *vs* tree outdegree with a depth of 6

Delays are more unstable especially for small bandwidths. Indeed, we performed 6 experimentations and obtained different random values at each run. This phenomenon is heavily

reflected by the the standard deviation on Figure 9 and Figure 10. It appears in the network trace that no TCP retransmission occurs during the file transfer. Random latencies may reflect artefacts in guests operating system (i.e., system calls, context switches, etc.) and requires further investigations. With *NEmu*, we have observed that a tree-shaped overlay of TCP connections has predictable data rates but unpredictable delays.

## IV. Related Work

*GNS* [13] is an open source software which allows to build a virtualized network topology. However, it is really close to CISCO systems because it manages *dynamips* emulators [14]. It can manage some QEMU virtual machines but with an alternative version of QEMU. *GNS* does not provide the possibility to build a community network. Finally *GNS* is hardly usable without any graphical interface making difficult the creation of a complex network. *Velnet* [15] is a virtual environment dedicated to teaching which uses VMware virtual machines. The complete topology can only run on a single host which implies strong limitations on the size of the virtual network. *ModelNet* [16] emulates a distributed virtual network but this one remains static at runtime. Thus, the dynamicity is not ensured with ModelNet. Further, the management of this system is fully centralized on an unique physical machine which disables the community aspect. *Vagrant* [17] uses *VirtualBox* virtual machines in order to emulate virtual network. The topology is hosted on a single physical machine and remains static at runtime. *Vnet* [18] is a distributed system which allows to connect several virtual machines hosted on different physical machine through the IP network. Even if it allows to create a distributed virtual network, this system only provides the interconnections. Therefore, no router or internal services are provided contrary to *NEmu*. Worldwide research testbeds such as PlanetLab, GENI and FEDERICA are quite different from *NEmu* because they are providing virtual infrastructures defined as *slices* of third parties owned hardware. Thus, the user needs a specific account, must comply to a use policy and has to use the tools, services and APIs of these testbeds.

Table I exhibits several properties of those previous solutions compared to *NEmu*. We see that *NEmu* can cover all usages (test and proof, performance evaluation as well as learn and teach). It can achieve a high realism as a research tool by managing dynamic virtual networks and by being able to be distributed over several physical machines. Furthermore, it can be easily used and deployed as no special rights are required on the physical machines. Finally, it offers a new feature called *community aspect* that enables several users to merge their virtual networks together in order to build a single larger network.

## V. Conclusion

*NEmu* is a tool for the creation and management of dynamic and heterogeneous virtual networks. Such virtual networks can be distributed over several physical hosts and be controlled without any administrative rights. We have shown how *NEmu* can be used to emulate, test and evaluate a host-based overlay network. The experimentation results of a file distribution application have shown that a media distribution tree made of TCP connections between the hosts can be used efficiently for non real-time applications but not for real-time ones thus confirming the results of previous studies [8] [9] and the validity of our testbed. These results demonstrate that *NEmu* is well suited for carrying out relevant performance evaluations upon overlay applications. Several next steps are already planned for our future work on *NEmu*. They consist in the following tasks by order of priority:

- The integration of migration capabilities for *virtual nodes* in order to handle load balancing,
- The integration of new `VRouter` services,
- The implementation of a GUI front-end.

## References


[1] B. Yamini and D. Selvi, "Cloud virtualization: A potential way to reduce global warming," in *RSTSCC*, nov. 2010, pp. 55–57.

[2] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proc. of the USENIX Annual Technical Conference*, 2005, pp. 41–46.

[3] N. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *Communications Magazine, IEEE*, vol. 47, no. 7, pp. 20–26, 2009.

[4] KVM, *Virtio*, http://www.linux-kvm.org/page/Virtio.

[5] NBD, *Network Block Device*, http://nbd.sourceforge.net.

[6] B. Hubert, G. Maxwell, R. Van Mook, M. Van Oosterhout, P. Schroeder, and J. Spaans, "Linux advanced routing & traffic control," in *Ottawa Linux Symposium*, 2003, pp. 213–222.

[7] R. Shingledecker, *TinyCore Linux*, http://distro.ibiblio.org/tinycorelinux.

[8] G. Kwon, J. Byers *et al.*, "Roma: Reliable overlay multicast with loosely coupled tcp connections," in *Proc. of the 23th IEEE INFOCOM*, 2004.

[9] G. Urvoy-Keller and E. Biersack, "A congestion control model for multicast overlay networks and its performance," in *Proc. of the 4th International Workshop on NGC*, 2002.

[10] P. Borgnat, G. Dewaele, K. Fukuda, P. Abry, and K. Cho, "Seven Years and One Day: Sketching the Evolution of Internet Traffic," in *Proc. of the 28th IEEE INFOCOM 2009*, April 2009, pp. 711–719.

[11] G. Dewaele, Y. Himura, P. Borgnat, K. Fukuda, P. Abry, O. Michel, J.J., R. Fontugne, K. Cho, and H. Esaki, "Unsupervised host behavior classification from connection patterns," *International Journal of Network Management*, vol. 20, pp. 317–337, 2010.

[12] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques," *ACM PAM*, pp. 205–214, 2004.

[13] GNS3, *Graphical Network Simulator*, http://www.gns3.net.

[14] C. Fillot, *Dynamips*, http://www.ipflow.utc.fr/dynamips/.

[15] B. Kneale, A. Y. De Horta, and I. Box, "Velnet: virtual environment for learning networking," in *Proc. of the 6th Australasian Conference on Computing Education*, vol. 30, 2004, pp. 161–168.

[16] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker, "Scalability and accuracy in a large-scale network emulator," in *The 5th OSDI*, 2002, pp. 271–284.

[17] M. Hashimoto and J. Bender, *Vagrant*, http://vagrantup.com.

[18] A. I. Sundararaj, A. Gupta, and P. A. Dinda, "Dynamic topology adaptation of virtual networks of virtual machines," in *Proc. of the 7th LCR Workshop*, 2004, pp. 1–8.


Table I
Comparison of Network Virtualization Tools

| Tool | Test & proof | Perf. eval. | Learn & teach | Dyn. net. | Distributed | Community | Spec. rights |
|---|---|---|---|---|---|---|---|
| GNS3 | yes | no | yes | no | yes | no | no |
| ModelNet | yes | yes | no | no | yes | no | no |
| *NEmu* | yes | yes | yes | yes | yes | yes | no |
| PlanetLab | yes | yes | no | yes | yes | yes | yes |
| Vagrant | yes | no | no | no | no | no | no |
| Velnet | no | no | yes | no | no | no | no |