

Precursors: Emulab

Robert Ricci and the Emulab Team

Abstract One of the precursors of the GENI project is Emulab, a testbed effort that has been ongoing at the University of Utah since 1999. Emulab is both the name of a *testbed control system*, and the name of a *particular facility* built using that system. The Emulab facility is housed at the University of Utah, but is available to researchers worldwide—thousands of users have run hundreds of thousands of experiments over the lifetime of the testbed. The Emulab software is open-source, and has been used to bring up dozens of experimental facilities at institutions around the world. Some of these, like the Utah facility, are open to the public for the purposes of research and educations; others are run by individual institutions for their own use, which may include product R&D, classified work, etc.

One of the precursors of the GENI project is Emulab, a testbed effort that has been ongoing at the University of Utah since 1999. Emulab is both the name of a *testbed control system* [29], and the name of a *particular facility* [10] built using that system. The Emulab facility is housed at the University of Utah, but is available to researchers worldwide [12]—thousands of users have run hundreds of thousands of experiments over the lifetime of the testbed. The Emulab software is open-source, and has been used to bring up dozens of experimental facilities at institutions around the world [11]. Some of these, like the Utah facility, are open to the public for the

The Emulab project was founded by Jay Lepreau, who led it from 1999 until his death due to cancer in 2008. As of 2016, the Emulab team includes: Keith Downie, Jonathon Duerig, Dmitry Duplyakin, Eric Eide, David Johnson, Mike Hibler, Dan Reading, Leigh Stoller, Kirk Webb, and Gary Wong. Over the last 16 years, dozens of people have worked on Emulab, including Christopher Alfeld, David G Andersen, David Anderson, Kevin Atkinson, Grant Ayers, Chad Barb, Srikanth Chikkulapelly, Steve Clawson, Austin Clements, Cody Cutler, Russ Fish, Daniel Montralloy Flickinger, Daniel Gebhardt, Shashi Guruprasad, Fabien Hermenier, Ryan Jackson, Abhijeet Joglekar, Xing Lin, Nikhil Mishrikoti, Ian Murdock, Yathindra Naik, Mac Newbold, Tarun Prabhu, Raghuv eer Pullakandam, Prashanth Radhakrishnan, Srikanth Raju, Pramod Sanaga, Timothy Stack, Matt Strum, Weibin Sun, Kevin Tew, Brian White, and Kristin Wright.

R. Ricci (✉)

Flux Research Group, School of Computing, University of Utah, Salt Lake City, UT, USA
e-mail: ricci@cs.utah.edu



Fig. 1 Early Emulab builders. *Left to right:* Mac Newbold, Logan Axson, Christopher Alfeld, Kristin Wright, Jay Lepreau, Mike Hibler

purposes of research and educations; others are run by individual institutions for their own use, which may include product R&D, classified work, etc.

The Emulab system was originally developed as the “Utah Network Testbed” by the Flux Research Group under the direction of Jay Lepreau. The facility’s initial purpose was to solve a problem that the group itself had: research in the area of computer systems (including networks) requires a great deal of hands-on experimentation, and performing those experiments necessarily means managing the equipment on which they will be run. It was clear, however, that the needs of the Flux group with respect to infrastructure were by no means unique, and the group decided to open the testbed to the wider research community in early 2000. Similarly, after several years of operating one facility, it became clear that others would like to run their own, similar infrastructure, and the software was generalized to run at other sites, with the first two being at the University of Kentucky [16] and Georgia Tech. Since then, the Emulab software has become the basis for a number of testbeds with different focuses, including: NSF’s CloudLab [23] (cloud computing), GENI [2, 3] (federation), PhantomNet [28] (mobile networking), PROBE [17] (large scale systems) and Apt [24] (adaptability and repeatability); DARPA’s National Cyber Range (security); and DHS’s DETERLAB [8] (security). The Emulab facility and codebase are key parts of the nationwide GENI infrastructure and several international federations in Europe, Brazil, Japan, and South Korea.



Fig. 2 Wiring in one of the Utah Emulab clusters

As its name suggests, Emulab was originally designed with emulation as its primary purpose: by this, we mean running “real” code on real hosts, interacting with a network that is “real” in the sense that it is constructed of real hardware such as switches and NICs, but which may be artificially manipulated in order to create effects such as latency, limited delay, etc. in a controllable manner. While emulation remains one of Emulab’s key use cases, it has grown far beyond this original focus, and supports other environments such as simulation, live wide-area network experimentation, wireless networks, and more.

Throughout its lifetime, the development of Emulab has been supported primarily by the National Science Foundation, with additional support from DARPA. Additional vendor contributions have come from Cisco, Intel, Compaq, Microsoft Research, Novel, Nortel, and HP. As a result of this support, users have never paid to use the Emulab facility at Utah or for using the source code to build other facilities. Emulab has also benefited greatly from the support of the University of Utah, in the form of machine room space, power, and cooling, as well as support from the IT organization for its unusual network needs and usage patterns. Emulab has never “stood still,” with each successive grant being used to add new, unique features such as wireless experimentation, support for virtualization, and federation.

1 Running Experiments on Emulab

To run an experiment on Emulab, a user describes the network he or she would like to run an experiment on. This specification is written in a dialect of the language that is used for the *ns-2* simulator [26], and is thus typically called an “NS file.” An example of such a file can be seen in Fig. 3. An NS file includes specifications of the nodes, links, and events that define the experiment. The user submits this NS file to Emulab, which attempts to find a free set of resources that match what is requested. For the most part, users do not ask for *specific* physical machines, and repeated runs of the same experiment might map onto a different set of physical hosts.

For nodes, the specifications of interest include the type of hardware to run on, the operating system to run, and possibly additional software packages to install. Emulab offers some flexibility in the specification of hardware: many users simply ask for a “pc,” which gives Emulab the flexibility to assign any one of the numerous hardware types it might control. Or, users can be more specific, referring to a specific type, if it has features that they require, or if they are trying to run on the same type repeatedly for consistency. Operating systems and software are often specified in terms of *disk images*, which Emulab automatically loads onto the hosts using a custom scalable multicast protocol. Emulab itself provides standard disk images for several Linux distributions, FreeBSD, and Windows, and users may also make their own (usually by customizing one of the facility images.)

In terms of links, the user may specify either point-to-point links or multipoint LANs, creating a *topology*. One of Emulab’s key features is that these links can

```

1  # Import definitions of Emulab-specific commands; Emulab also provides a
2  # stubbed-out version of this file for use inside of the ns-2 simulator
3  source tb_compat.tcl
4
5  # The Simulator object, from ns-2, encapsulates the representation of the
6  # topology
7  set ns [new Simulator]
8
9  # Scripts can include variables that can be modified to alter the topology or
10 # behavior of an experiment
11 set num-pcs 16
12
13 set lan-string ""
14
15 # Full OTcl language features, such as loops and string manipulation, are
16 # available
17 for {set i 1} {$i <= $num-pcs} {incr i} {
18
19     # Create a new node, requesting a specific operating system
20     # and type of hardware
21     set node($i) [$ns node]
22     tb-set-node-os $node($i) FreeBSD-STD
23     tb-set-hardware $node($i) pc3000
24     append lan-string "$pc{$i}_"
25
26 }
27
28 # Put all nodes into a 1Gbps LAN with no traffic shaping
29 set lan0 [$ns make-lan "$lan-string" 1000Mb 0ms DropTail]
30
31 # Indicate the the topology definition is complete (in ns-2, this would start
32 # the simulation)
33 $ns run

```

Fig. 3 An example of an “NS file” used to describe a topology in Emulab

have traffic shaping parameters attached to them: bandwidth, base latency (typically modeling propagation delay), base packet loss (before congestive losses), and queuing discipline. These enable a cluster that resides entirely within a single datacenter to emulate networks that cross large distances. Typically, experimenters set these to values that are representative of real networks, but one of the strengths of Emulab is that they can be set to *any* value (within the limits of testbed hardware), enabling sensitivity analyses, “what if” experiments, etc. Links in the request are implemented in Emulab using VLANs, and if any traffic shaping parameters are specified, a node running the Dummynet [4] emulator is transparently inserted into the middle of the link to realize them.

Most experimenters use Emulab interactively; that is once their nodes are ready, they log in via `ssh` and use the command line to launch their experiments. However, they may also specify a set of *events* in the NS file. These events can run at specified to run at certain times (relative to the successful reservation of resources), or be grouped into *timelines* that can be launched on-demand at any point during the experiment. Events can run programs, modify traffic shaping parameters, take nodes or links up or down, and more. Experiments that are specified in this way can be run as *batch* experiments, in which Emulab waits for sufficient resources to become available, instantiates the experiments, and lets the events run to completion.

Once the user has submitted an NS file, Emulab *swaps in* (a term borrowed from virtual memory) that experiment on physical hardware. This process typically takes a few minutes; time varies depending on the number of nodes in the experiment, whether custom disk images need to be loaded, etc. Emulab *maps* the requested topology onto the resources available at the time: this means finding nodes that match the user’s specification, and finding paths across the physical network that meet the requirements of the requested links. In general, users do not expect to get the same nodes for different runs of the same experiment, but can request specific *types* of nodes to get some assurance that successive experiments are run on machines that are effectively identical for most purposes. Experiments typically last hours to days; Emulab encourages users to hold resources for only as long as they are actively using them. When a user is done with a particular allocation of resources, he or she can either *swap out* the experiment or *terminate* it. Swapping out preserves the experiment definition from the NS file (but not node, storage, or network state) so that it can easily be swapped back in later.

2 The Emulab Control Infrastructure

The Emulab control infrastructure (shown in Fig. 4) consists of three kinds of hosts: one *boss* node, one *ops* node, and zero or more *sub-boss* nodes. (In recent installations, *boss* and *ops* are typically run in two VMs on one physical host.) The collection of hardware controlled in a typical Emulab installation includes *nodes*, one or more *switches*, and devices to provide management control over the nodes (power controllers, serial console concentrators, etc.) Emulab controls

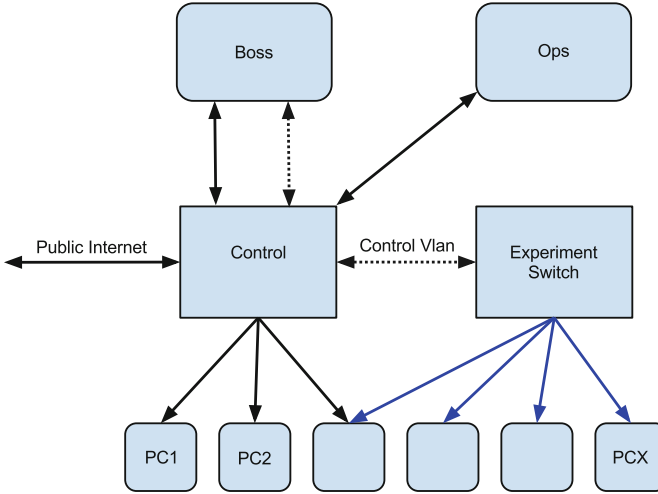


Fig. 4 Emulab’s control infrastructure

nodes by controlling their boot process and through disk imaging [7, 14, 20]; it control switches by configuring them through SNMP, NETCONF, and other network configuration protocols.

The **boss** node hosts most of Emulab’s critical control infrastructure: the database, webserver, DNS server, various boot and imaging servers, etc. Many infrastructure control interfaces, such as power controllers, switches’ SNMP interfaces, etc. are protected by VLANs, so that they can only be controlled from **boss**. Due to its sensitive nature, users are not given shell access to **boss**. The primary interactions that **boss** has with nodes and switches occur during experiments swap in and swap out, as **boss** manages the boot and image loading processes of the nodes, and it is during the swap process that **boss** configures switches. While the resource needs for a **boss** server are not extreme, it can become the bottleneck for instantiating large experiments. For large installations (more than two or three hundred nodes), some work can be offloaded to sub-bosses, which can handle boot and imaging services for a subset of the cluster. sub-bosses provide only read-only services (such as answering DHCP queries and distributing disk images); this design allows us to avoid complicated state synchronizations between the various **boss** and sub-**boss** nodes.

The main functions of **ops** are to give users a place to get a shell independent of particular experiments and to act as a central fileserver. In some Emulab installations that do not have public IP addresses, **ops** acts as a sort of a bastion host, where users must log in before they can reach the (private) control network interfaces of the nodes in their experiments. As discussed later, we discourage users from over-reliance of the network filesystem hosted on **ops**, but it does serve useful functions with respect to user home directories, etc.

3 Distinguishing Features of Emulab

Emulab is far from the only system for managing collections of servers. In recent years, cloud management systems such as OpenStack [27] and Eucalyptus [18] have become popular ways to manage virtualized resources. More contemporaneous with Emulab's initial conception, various cluster management toolkits such as ROCKS [25], xCat [9], and Grid [13] systems have a long history as well. In the research infrastructure space, management systems such as PlanetLab [6], the ORBIT Management Framework [19], and ORCA [5] are also used to run testbeds.

What makes Emulab unique among these systems is the emphasis it places on three things: scientific fidelity, bare-metal resource provisioning, and the network as a first-class concern. This has led Emulab to make a number of different design decisions, in terms of the makeup of its physical infrastructure, the architecture of its control software, and the way that users interact with its resources.

3.1 *Focus on Scientific Fidelity*

One of Emulab's design goals from the beginning has been a focus on scientific fidelity. This means that, to the extent possible, experiments run on Emulab should not contain artifacts that are the result of other concurrent use of the facility. This is quite different from the goal of satisfying service-level agreements (SLAs), as clouds aim to do, and leads to a number of properties that distinguish it from other datacenter or infrastructure-as-a-service offerings.

The first consequence of this philosophy is an emphasis on allocation and provisioning of resources at a "bare-metal" level. Most experiments are run on nodes that are completely dedicated to a single experiment at a time. While this is more challenging to provision than a virtualized environment or one with shared hosts, it means that processes belonging to different users do not compete for CPU time, memory, I/O bandwidth, etc. It also gives users direct access to hardware and full control over the operating system, features that are critical to researchers in operating systems and networks.

Second, all Emulab nodes are connected to two networks: one "control network" and one "experiment network" [29]. The control network is shared by all experiments at once, and it is connected to the public Internet. It is over this network that users log into nodes, that remote filesystems are mounted, and users are encouraged to run traffic that coordinates their experiment, collects logs, etc. Emulab makes an attempt to provide isolated performance on the control network. The experiment network is isolated: it is configured to match the topology requested by the experimenter, and experiments do not see each others' traffic on this network. Experiments may run whatever protocols they wish, at whatever speed they would like, on this network. While cost constraints prevent this network from having full bisection bandwidth, one of the key features of Emulab's resource mapper is that it

uses information about the topology submitted by the user to find an embedding that minimizes use of bottleneck links in the physical switching topology [21]. Thus, experimenters can have confidence that they are competing only minimally with other users on this network.

Third, the focus on fidelity affects Emulab’s strategy towards storage. Many clusters make heavy use of shared, network-mounted filesystems. While this is convenient for users, it is less desirable from a fidelity standpoint: any experiments whose behavior is affected by the performance of filesystem I/O become vulnerable to interference from other users. Thus, while Emulab does offer some shared filesystems, they are relatively small and low-performance, and users are strongly encouraged to rely on the local disk of each node for their experiments. Each physical node has one or more local disks, and like the nodes themselves, they are not shared with any other simultaneous experiments. This does make persisting large datasets across experiments more difficult, but Emulab offers features for easy disk image creation, which helps alleviate this problem for many users.

Finally, it is critically important that Emulab provide users with the resources having the performance that the user requested: if an experiment is un-knowingly run under the wrong network conditions, it can produce an incorrect result. These kinds of events have occurred in Emulab in the past, due to mis-wired networks, bugs in the provisioning software, etc. Thus, Emulab incorporates a piece of software called “linktest” [1] which is run on new experiments. Linktest has the job of ensuring that the topology that Emulab has instantiated is the same as that requested by the experimenter, and that any link shaping parameters (delay, bandwidth, etc.) are within set tolerances of what was requested. Linktest takes a completely separate code path from the standard Emulab provisioning software to guard against bugs in the Emulab software—it even goes so far as to use a completely separate parser for NS files. The result is a high assurance that the user has received the network that they requested.

3.2 Focus on Multi-Tenant, Bare-Metal Allocation

Emulab has always focused primarily on the provisioning of bare-metal, rather than virtualized, resources. It has, over time, added the ability to provision virtual machines and container-based operating systems [15], but its core strength continues to lie in bare metal. This is for three reasons: first, as discussed above, this provides an environment with higher fidelity and lower probability of artifacts due to shared use. Second, Emulab’s original use case (and one of its major uses still) is for development of low-level code such as operating system kernels, hypervisors, software switches, and other code that requires direct hardware access. Third, Emulab development began before the current generation of virtualization technologies was mature.

This means that the provisioning systems in Emulab are focused on controlling the booting of physical machines. Emulab retains control by booting all nodes off

of the network (and not allowing users to change this setting.) A simple, small bootloader checks to see what it should do next: this might mean continuing to boot from the disk, loading a kernel from the network for disk imaging, etc. Emulab “cleans” machines between users by loading a default disk image every time a node is released from an experiment. It does *not* attempt a secure erase of the entire disk, as this would be extremely time consuming and is overkill for most research use. One interesting aspect of this process is that, for security reasons, Emulab needs to be sure that its own disk loader program has been run and has loaded the correct image, rather than malware impersonating the disk loader, leaving backdoors, etc. behind. To this end, we have developed the Trusted Disk Loading System (TDLS) that uses the Trusted Platform Module (TPM) to attest to the secure booting of the disk loader [7].

One consequence of the focus on bare metal is on the efficiency of allocation. Put simply, virtual allocation can be more efficient, because it shares hosts between multiple simultaneous tenants and can over-subscribe resources to take advantage of users who do not fully utilize their allocations. Virtual allocations can also be finer-grained. However, these properties would violate Emulab’s need for artifact-free experimentation. Combined with the fact that Emulab does not charge for use (providing no incentive for users to minimize their resource consumption), this means that Emulab must take measures to ensure that resources are used efficiently. The major strategy that Emulab employs in this regard is idle monitoring: the standard Emulab OS images (and thus, most user images that derive from them) include daemons that monitor for CPU and network activity, remote logins, etc. If an experiment goes idle, the user is notified, and has some set period (typically a few hours) to return to using it before resources are automatically reclaimed. Of course, it is possible for users to trick the idle detection system, for example by running programs that consume CPU cycles but do no useful work, but we have found that most users are good citizens and that this puts sufficient pressure on most of them to behave responsibly.

3.3 *The Network as a First-Class Entity*

Because Emulab began life with an emphasis on network experimentation, it has always viewed the network as a first-class entity; whereas many cluster management systems attempt to abstract over the network, or view network setup as a side effect of provisioning compute resources, Emulab gives the network topology equal importance. One consequence of this emphasis is that hosts, storage, and networks are specified *together* in Emulab NS files. This means that Emulab is less suited to the “elastic” provisioning style of clouds, but excels at uses cases in which one needs to capture a description of the whole environment.

Emulab’s topology specification language is designed to handle both large broadcast domains (eg. LANs) and point-to-point links, and is thus amenable to complex topologies. In contrast to, say, cloud providers, who have no information

about the intended communication patterns of their tenants, Emulab knows exactly how much bandwidth it must provision, and where. Its network mapper, called *assign* [21], maps the links and LANs of the submitted topology on to available hardware, ensuring that every link in the submitted topology is mapped to sufficient capacity on the network.

Emulab uses simple VLANs to segregate traffic on the experiment network, since they map directly to the abstraction provided to users and are available on all managed Ethernet switches. VLANs ensure that traffic from one experiment is not visible in others, enabling experimenters to use whatever addresses, protocols, etc. they wish within their experiments. Emulab does have some capabilities for exposing switch programming primitives, such as OpenFlow, but it views these as features offered to users rather than as mechanisms for provisioning. The Emulab facility at Utah also includes a set of switches that are allocatable to users in the same way that PCs are: for the duration of the experiment, the switch is reserved for the exclusive use of the experimenter and is under their full control. These switches are connected to a set of layer-1 switches, which perform simple forwarding without any Ethernet protocol processing, making them ideal for experiments that examine or modify the Ethernet layer.

4 The Evolution of Emulab into ProtoGENI

One of the key features that was historically lacking from the Emulab codebase was support for federation. Each of the “classic” Emulab clusters is an island unto itself; if an experimenter wants to use more than one, he or she must apply for accounts on each, and no support is provided for moving files, disk images, etc. between clusters.

This changed significantly with Emulab’s participation in the GENI project. For this project, Emulab evolved to become ProtoGENI [22]. The name stems from the fact that it was initially viewed prototype of GENI; this is now anachronistic, as Emulab now contains one of the most complete implementations of the GENI concepts and APIs.

The Emulab software and the ProtoGENI software are one and the same; in essence, ProtoGENI is an alternative interface to Emulab. It is a new set of APIs, specified by GENI, and the ecosystem of tools that exists on top of those APIs. Underneath, when these APIs are invoked, they are mapped onto existing internal Emulab features and concepts. For example, the GENI notion of “slices” [3] map on to Emulab “experiments.” Emulab does have its own set of APIs that pre-date the GENI project; however, they were never heavily used by users, and they did not include any notion of federated access.

ProtoGENI includes both a set of “native” ProtoGENI APIs and the “official” GENI APIs. The “native” APIs predate the standardization of the GENI APIs, and were among the many influences on the GENI standards. Thus, the two sets of APIs are similar, and offer similar feature sets, but are not completely identical.

ProtoGENI does choose not to expose some features from Emulab that are problematic for federation. For example, Emulab includes a shared filesystem that is available on all nodes. This works fine in a cluster, where all nodes have relatively fast and reliable access to the fileserver, but it becomes a major liability when it must be exported across the wide area, so ProtoGENI does not expose this feature. Similarly, Emulab includes a publish-subscribe event system that works smoothly within one cluster, but is hard to federate across the wide area; this, however is a feature we may attempt to re-introduce in the future.

The addition of the GENI/ProtoGENI features has opened a new era for Emulab; it used to be that each Emulab installation operated completely independently, and it is now possible to create portals that offer access to many of them at the same time. Also, while it used to be the case that almost all features in Emulab had to be offered by the Emulab software itself, the opening up of a richer set of APIs means that a greater number of tools, developed for GENI, can now be used on it.

5 Lessons from Emulab

Emulab has always taken the approach of making hardware and new software features available to users as soon as possible, even when those features are not yet complete, and/or there are still features planned that will add more “polish” or user-friendliness. This strategy has been highly successful; for most new features, there exist a set of “power users” who are the early adopters, and by their use, provide feedback on how future development should be directed. We have sometimes found that features that we thought were incomplete, in fact, already offer users everything they need, or have found that certain features were not as valuable as we had thought. The GENI project has adopted a similar approach with its “spiral” development.

It has been critical to develop a community of users around Emulab. Overall, the burden of user support for the facility is relatively light given the size of its userbase and the complexity of the features that it makes available. We attribute this in part to the fact that users within an institution or collaborative group tend to talk with each other, sharing stories of what has worked, not worked, workarounds for problems encountered, etc. The flipside of this lesson is that Emulab has put such an emphasis on user privacy that it has perhaps erred too much on the side of making it difficult for users to publicly share with each other. Emulab adopted a public user mailing list fairly late in its development, and sharing artifacts such as NS files across projects can be problematic as they often depend on private resources such as fileserver space. This is an issue we are working to address with ongoing development activities [24].

Emulab’s decision to use a full programming language (NS scripts, written in OTcl) as its topology specification language has been both a major asset and limiting factor. It is extremely useful to be able to use the constructs of a full programming language, such as loops for building regular network topologies, and conditionals for constructing scripts that are flexible and parameterizable. On the other hand, it means that these descriptions are not, themselves, directly machine-manipulable;

Emulab essentially “compiles” NS files that are submitted to it to an internal format, which users do not have direct access to. This means that generating NS files programmatically is possible (though awkward, as code generation often is), but that writing tools such as GUIs to manipulate them after they have been written is nearly impossible. Emulab does have a topology GUI, written in Java, but that GUI is only able to understand a very small subset of the language, and cannot be used to view or manipulate NS files that have loops, conditionals, etc.

Going forward, a way to keep the “best of both worlds” in terms of a rich, user-friendly topology specification is to have a well-defined declarative representation free of complex programming language constructs, but which (unlike Emulab’s internal representation) is exposed to users. This representation can be used as a “compiler target” for one (or more) higher-level languages that offer the flexibility and programmer-friendliness of ns-2. This is essentially the design choice that GENI has made: the low-level representation is the RSpec, which can be generated with `geni-lib`, which is a library for Python that (among other things) helps construct RSpecs.

It was also originally hoped that by using a language that came from a network simulator, we would make it easy for users to move back and forth between simulation and emulation. In practice, this has not materialized. There are many potential reasons for this: perhaps the communities that value these methodologies are non-overlapping, perhaps the topology description itself is not enough to ease the transition, or perhaps there is simply little call for such transitions. Regardless, in retrospect, picking ns-2 as a base language in Emulab was not a poor choice, but direct, native integration of emulation and simulation is not a design feature that Emulab is likely to pursue in the future.

We believe that, in large part, Emulab’s success can be attributed to the fact that the Flux Research Group built a facility designed around its own needs, and those needs were reasonably representative of a larger research community.

6 The Future of Emulab

One of the main themes of Emulab’s continued development is as a *platform* for the management of testbeds, clusters, clouds, and other facilities. The Emulab team cannot possibly, by itself, develop infrastructure that provides the features needed for the entire computer science research community; what it can do, however, is develop a base layer of hardware and software on top of which others can build infrastructure that meets their own communities’ needs. Because it provisions at a bare-metal level, and because it takes a holistic view of the network, storage, and other aspects of the facility, it is in a good position to have other types of infrastructure deployed on top, using (in GENI terms) “slices.” Additionally, because of the federation features it now has thanks to GENI, it is relatively easy to deploy such infrastructure across cluster and organizational boundaries.

The direction of Emulab's development is perhaps best typified by two follow-on testbeds built by the Flux Research Group: Apt [24] and CloudLab [23].

Apt, the Adaptable Profile-Driven Testbed, is a facility that is customizable with "profiles." A profile represents an encapsulation of the environment necessary to run a particular experiment, or a class of experiments. Each of these profiles can thus be thought of as a mini-testbed, running on top of Apt's hardware infrastructure. Profiles can be created by experts in particular domains, and shared with others. For example, an expert in database systems could install a standard set of database software, workload generators, reference datasets, etc., and share it with her or his community; likewise, someone from the high-performance computing community could do the same for domain science MPI-based code. Indeed, the Apt cluster is shared between computer scientist and the University of Utah's Center for High Performance Computing, with nodes being moved back and forth between the two sides on demand. Profiles can be shared either as semi-permanent facilities that many users access simultaneously, or their definitions can be shared, via hyperlink, such that every user gets their own instantiation, as happens with Emulab experiments. This is a powerful way to share research code and data; for example, the authors of a paper can create a profile that encapsulates everything needed to repeat the experiments in their paper, and provide this link in the paper itself or on their website.

CloudLab is a facility built for researchers who need their "own cloud" in order to conduct their work. While public and private clouds are extremely useful for many types of research, they have major limitations when it comes to transparency and control of software and resources at the bottom of the stack. There are certain elements that, by their nature, are considered part of the infrastructure, and cannot be changed (or even, typically, observed) by users; these include the hypervisor, the network, and the storage systems. For researchers who want to study and improve those parts of the stack, simply using "someone else's" cloud is not enough. Using Emulab for provisioning and federation, CloudLab is a facility where researchers can have exactly this sort of access. Like a real cloud, CloudLab is physically distributed, with sites at the University of Utah, the University of Wisconsin Madison, and Clemson University. Each of these clusters is an autonomous Emulab instance, and they are federated with each other and with GENI. So far, the infrastructure is a great success, with over 800 users running experiments in its first year of operation.

References

1. Anderson, D.S., Stoller, L., Hibler, M., Stack, T., Lepreau, J.: Automatic online validation of network configuration in the Emulab network testbed. In: Proceedings of the Third IEEE International Conference on Autonomic Computing (ICAC 2006) (2006)
2. Bastin, N., Bavier, A., Blaine, J., Chen, J., Krishnan, N., Mambretti, J., McGeer, R., Ricci, R., Watts, N.: The InstaGENI initiative: an architecture for distributed systems and advanced programmable networks. *Comput. Netw.* **61**, 24–38 (2014)

3. Berman, M., Chase, J.S., Landweber, L., Nakao, A., Ott, M., Raychaudhuri, D., Ricci, R., Sesar, I.: GENI: a federated testbed for innovative network experiments. *Comput. Netw.* **61**, 5–23 (2014)
4. Carbone, M., Rizzo, L.: Dummynet revisited. *ACM SIGCOMM Comput. Commun. Rev.* **40**(2), 12–20 (2010)
5. Chase, J., Grit, L., Irwin, D., Marupadi, V., Shivam, P., Yumerefendi, A.: Beyond virtual data centers: toward an open resource control architecture. In: *International Conference on the Virtual Computing Initiative (ICVCI)* (2009)
6. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: PlanetLab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Comput. Commun. Rev.* **33**(3), 3–12 (2003)
7. Cutler, C., Hibler, M., Eide, E., Ricci, R.: Trusted disk loading in the Emulab network testbed. In: *Proceedings of the Third Workshop on Cyber Security Experimentation and Test (CSET)* (2010)
8. DeterLab: Cyber-security experimentation and testing facility (web site). Information Sciences Institute, University of Southern California. <http://www.deterlab.net> (2016). Accessed Jan 2016
9. Extreme Cluster/Cloud Administration Toolkit. <http://www.xcat.org> (2016). Accessed Jan 2016
10. Emulab.net: Network emulation testbed web site. Flux Research Group, School of Computing, University of Utah. <http://www.emulab.net> (2016). Accessed Jan 2016
11. Emulab.net: Other Emulab testbeds. Flux Research Group, School of Computing, University of Utah. <https://wiki.emulab.net/Emulab/wiki/OtherEmulabs> (2016). Accessed Jan 2016
12. Emulab.net: Projects that have actively used emulab.net. Flux Research Group, School of Computing, University of Utah. <http://www.emulab.net/projectlist.php3> (2016). Accessed Jan 2016
13. Foster, I., Kesselman, C.: *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco (1999)
14. Hibler, M., Stoller, L., Lepreau, J., Ricci, R., Barb, C.: Fast, scalable disk imaging with frisbee. In: *Proceedings of the USENIX Annual Technical Conference*. USENIX (2003)
15. Hibler, M., Ricci, R., Stoller, L., Duerig, J., Guruprasad, S., Stack, T., Webb, K., Lepreau, J.: Large-scale virtualization in the Emulab network testbed. In: *Proceedings of the USENIX Annual Technical Conference* (2008)
16. Laverell, W.D., Fei, Z., Griffioen, J.N.: Isn't it time you had an Emulab? In: *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (2008)
17. NMC Probe (Web site). <http://www.nmc-probe.org> (2016). Accessed Jan 2016
18. Nurmi, D., Wolski, R., Grzegorzczak, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The eucalyptus open-source cloud-computing system. In: *IEEE/ACM International Symposium on Cluster Computing at the Grid* (2009)
19. Ott, M., Sesar, I., Siracusa, R., Singh, M.: ORBIT testbed software architecture: supporting experiments as a service. In: *Proceeding of IEEE Tridentcom* (2005)
20. Ricci, R., Duerig, J.: Securing the Frisbee multicast disk loader. In: *Proceedings of the First Workshop on Cyber Security and Test (CSET)* (2008)
21. Ricci, R., Alfeld, C., Lepreau, J.: A solver for the network testbed mapping problem. *ACM SIGCOMM Comput. Commun. Rev.* **33**(2), 65–81 (2003)
22. Ricci, R., Duerig, J., Stoller, L., Wong, G., Chikkulapelly, S., Seok, W.: Designing a federated testbed as a distributed system. In: *Proceedings of the 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom)* (2012)
23. Ricci, R., Eide, E., The CloudLab Team.: Introducing CloudLab: scientific infrastructure for advancing cloud architectures and applications. *USENIX ;login:* **39**(6), 36–38 (2014)
24. Ricci, R., Wong, G., Stoller, L., Webb, K., Duerig, J., Downie, K., Hibler, M.: Apt: A platform for repeatable research in computer science. *ACM SIGOPS Oper. Syst. Rev.* **49**(1), 62–69 (2015)
25. Rocks Cluster Distribution. <http://www.rocksclusters.org> (2016). Accessed Jan 2016

26. The NS-2 User Manual. <http://www.isi.edu/nsnam/ns/> (2016). Accessed Jan 2016
27. The OpenStack Website. <http://www.openstack.org> (2016). Accessed Jan 2016
28. The PhantomNet Testbed. <http://www.phantomnet.org> (2016). Accessed Jan 2016
29. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. In: Proceedings of the USENIX Symposium on Operating System Design and Implementation (OSDI). USENIX (2002)

The GENI Book

McGeer, R.; Berman, M.; Elliott, C.; Ricci, R. (Eds.)

2016, XXX, 651 p. 255 illus., 225 illus. in color.,

Hardcover

ISBN: 978-3-319-33767-8