Scaling Network Emulation with Multiplexed Virtual Resources

Shashi Guruprasad, Leigh Stoller, Mike Hibler, Jay Lepreau

School of Computing, University of Utah

www.flux.utah.edu www.emulab.net

To avoid experimental artifacts, the original Emulab network emulation system used conservative resource allocation. It mapped virtual network nodes and links one-to-one onto dedicated PCs and switched Ethernet links. We have three motivations for relaxing this constraint, allowing controlled multiplexing of virtual onto physical resources. First, some applications such as p2p systems require large topologies for evaluation, yet are not resource-hungry. Second, much research and educational use does not need perfect performance fidelity, or does not need it on every run. Third, multiplexing makes small-scale emulation clusters much more useful.

Our first goal is to design mechanisms for virtual nodes ("vnodes"), links ("vlinks") and LANs that make them as similar to their real-life counterparts as possible, while efficiently using physical resources. Classic VMs such as VMware are the most similar to real machines, but are relatively inefficient. Unix processes provide a lightweight vnode, but are least like a PC and would require application code to be modified. Other options include User-mode Linux, Xen VM's "paravirtualization," and our design, which virtualizes a host's process, network and filesystem (FS) namespaces without compromising performance. We achieve our second goal, transparency: in the new Emulab, vnodes are almost indistinguishable from PCs in terms of specification, access, and API. Experimenters are able to login to these vnodes and run unmodified programs.

This system provides a new point on the spectrum between simulation, in which all resources are virtual, and emulation, in which all resources are physical. In supporting these multiplexed resources, we take advantage of Emulab's strengths, especially resource mapping, node management, and node configuration. Relieving the experimenter of these tasks is essential in making large experiments feasible. Integrating automation and virtualized nodes/links into a single system in which the nodes are completely programmable, makes Emulab, to our knowledge, the most complete system extant for controlled network experimentation.

The key issues, some still open, are virtual nodes and links/LANs, routing, performance isolation, mapping, and scaling. We outline some aspects of these below.

Virtual nodes: BSD *jails* provide our starting point for virtualizing nodes, restricting a process and all its descendents to a unique slice of the FS, network, and process namespaces. However, they do not provide everything necessary for network experimentation. We enhanced the network aspects of jails in several ways; to a large extent, we could have instead merged Zec's cloneable network stack work. For example, packet forwarding is best performed in the kernel on behalf of vnodes, for performance reasons. We therefore added independent routing tables to the kernel, derived from Scandariato's VPN work. During packet forwarding, we preserve the context of a vlink on which a packet arrived so that route lookups are directed to the correct table.

In order to constrain disk usage, we use per-vnode virtual disks ("vdisks") rather than different subtrees of the same FS. We implement this by using the BSD *vn* disk driver that lets a regular fixed-size file be exposed via the disk interface. There is one vdisk per vnode that contains files specific to that vnode and forms the root of its filesystem tree. Other filesystems are shared and are loopback-mounted into the jail FS namespace.

Virtual links and LANs: Multiplexing vlinks on physical links ("plinks") requires virtual network interfaces. All existing alternatives have drawbacks. IP-in-IP tunneling is point-to-point and cannot emulate LANs. 802.1Q VLAN trunking is limited to 4096 VLANs which constrains the scale and number of simultaneous experiments. Multiple pseudo-MAC addresses per interface is another approach, but requires interfaces to be in promiscuous mode and might overload MAC tables in switches, causing them to act as hubs. Instead, we developed a *virtual ethernet device* which is an unusual hybrid of a virtual device, an encapsulating device, and a bridging device. It allows us to create large numbers of ethernet interfaces (virtualization), multiplex them on physical interfaces or tie them together in a loopback fashion (bridging), and have them communicate transparently through our switch fabric (encapsulation). We support different speeds and latencies using the Dummynet traffic shaper.

Mapping: Vnodes must be partitioned across physical nodes ("pnodes") in order to realize the topology; Emulab performs this mapping automatically using combinatorial optimization. For non-trivial topologies, it is infeasible to do this partitioning manually, and simplistic partitioning can lead to network artifacts or poor use of resources. Emulab's mapping algorithm allocates link bandwidth conservatively, ensuring that plinks are not overused, and also attempts to use resources efficiently.

Current issues: *Performance isolation or monitoring* is required if emulation accuracy is a goal. We control network bandwidth via Dummynet and disk usage by vdisks. Currently, the experimenter controls the allocation of node CPU and memory by specifying an ad hoc "co-location factor." Memory has been more limiting than CPU; our observed limit is about 20 vnodes on a 512MB PIII. The *mapping* algorithm is a challenge, as it can find poor solutions for large virtualized topologies, and exhibits scaling problems. *Scaling* is a pervasive challenge, as the order of magnitude size increase from vnodes stresses many parts of the Emulab system. Besides improving centralized services, we will attack this problem by relaxing the transparency, to Emulab management software, of vnodes. Visibility into the underlying pnode allows many forms of optimization and tuning. Today, our largest automatically-configured experiment has 520 virtual nodes, which mapped to 44 PCs. It took 28.3 minutes to instantiate; the mapping program consumed 48% of that, on a 1Ghz PIII.

Sponsors: NSF ANI-0082493, ANI-0205702, Cisco, DARPA F30602-99-1-0503, F33615-00-C-1696. Email: testbed@flux.utah.edu.