

2003

vBET: A VM-Based Emulation Testbed

Xuxian Jiang

Dongyan Xu

Purdue University, dxu@cs.purdue.edu

Report Number:

03-014

Jiang, Xuxian and Xu, Dongyan, "vBET: A VM-Based Emulation Testbed" (2003). *Computer Science Technical Reports*. Paper 1563.
<http://docs.lib.purdue.edu/cstech/1563>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

vBET: A VM-BASED EMULATION TESTBED

**Xuxian Jiang
Dongyan Xu**

**Department of Computer Sciences
Purdue University
West Lafayette, IN 47907**

**CSD TR #03-014
April 2003**

vBET: a VM-Based Emulation Testbed

Xuxian Jiang
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA
jiangx@cs.purdue.edu

Dongyan Xu
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA
dxu@cs.purdue.edu

ABSTRACT

With the increasing requirement of robustness and predictability for network protocols and distributed systems, it becomes desirable to develop realistic, customizable, and scalable emulation testbeds for the testing and evaluation of network and distributed protocols. A number of recently proposed emulation testbeds have clearly demonstrated the advantage and promise of this approach. Meanwhile, more efforts are necessary to achieve higher degree of flexibility and reality, as well as customizability, such as stronger support for arbitrary topology setup and core node customization.

In this paper, we present vBET, a versatile and scalable emulation testbed based on the virtual machine technology. vBET is formed by one or more physical commodity servers, and is therefore readily and locally deployable in a research lab. vBET creates a virtual distributed environment with both network infrastructure and end systems. Each entity, such as a router, a switch, a firewall, or an application-level proxy, is emulated by a virtual machine running unmodified system or application software. Furthermore, the entities emulated by vBET are user-configurable and can be deployed on-demand. The same vBET (physical) server can be easily configured and setup as the testbed for different purposes, such as Internet routing, distributed firewalls, or peer-to-peer networks.

We describe the implementation and application of vBET. For the implementation, we present key enabling techniques including virtual OS, virtual topology, small-footprint file system, as well as a network topology modeling language. For the application of vBET, we present the creation of different experiment environments using vBET, including OSPF routing, distributed firewall, and Chord peer-to-peer network. These experiments demonstrate the versatility, customizability, efficiency and scalability of vBET.

1. INTRODUCTION

There has been increasing requirement of robustness and predictability for network protocols and distributed systems, such as IP routing [18, 21] and packet scheduling [23], peer-to-peer systems [8, 10, 17], overlay networks [6, 5], and the computation/data grid [11, 7]. It has become desirable to develop realistic, customizable, and scalable emulation testbeds for the testing and evaluation of these protocols and systems.

Meanwhile, traditional simulation tools, such as the widely used ns-2 [1], are more available and economical, due to their easy installation, management, and relatively low resource requirements. Unfortunately, simulation-based experiments may be deemed less convincing, due to their lack of fidelity to real-world environments. On the other hand, real-world experiments are based on realistic settings and therefore more credible. However, they are highly complex and costly to set up, control and monitor. Between the two ends, emulation provides a good trade-off between fidelity and cost. The goal (and challenge) of emulation testbed development is therefore to achieve controllability, configurability, reproducibility, ease of setup and management, and scalability. In particular, the emulation testbed should be *flexible* enough to create arbitrary network infrastructure and topology for different experiments.

Recently, a number of real-world or emulation testbeds have been successfully deployed. Representative testbeds include PlanetLab[20], Netbed[26], and ModelNet[25]. These testbeds clearly demonstrate the advantage and promise of emulation-based experimentation. However, there still exist a number of challenges which are yet to be addressed. In this paper, we consider the following requirements for the design and implementation of an effective emulation testbed.

- *Easy and wide deployment* The testbed is expected to be easily and locally deployable in any research lab equipped with commodity servers or high-end desktops. Therefore, researchers will have full control over the testbed and enjoy more convenient execution and monitoring of the experiments. None of PlanetLab, Netbed, and ModelNet can be readily deployed without the availability of substantial hardware resources.
- *Setup of arbitrary network topology* The testbed should be able to accommodate as many distributed systems as possible to maximize testbed utilization. But once

devices are procured and testbed is physically deployed, some limitations are also created, such as the number of ports available in one particular switch or router as well as the same routing code always running on the same devices. Furthermore, it is cumbersome and not easy to re-wire the physical connection among the testbed¹. There is a need for stronger virtualization of network connection in order to set up flexible network topology.

- *Customization of core nodes* Most current emulation or real systems, such as PlanetLab[20], Netbed[26], ModelNet[25], are capable of customizing the end systems. Unfortunately, they provide rather limited support for the customization of core nodes of the system, such as replacing the packet queuing discipline or employing a different route lookup algorithm in one particular router. It is also desirable to completely change the roles of core nodes in different experiments - for example, a core node may be a high-performance router in one experiment setup, and an advanced stateful firewall in another experiment.
- *Fast setup of experimental topology* Most current available emulation testbeds, except Netbed[26], lack the capability of fast setup of experimental network topology. It is expected that such setup be automatically performed within *seconds*, so that the researchers do not have to experience long waiting time during the experiments.
- *Coordination between core nodes* In current testbeds, it is cumbersome to coordinate between core nodes, due to the constraints on physical connection of nodes. Software or dedicated physical wire [26] is used for the coordination between core nodes. As a result, distributed systems, such as distributed firewall or distributed reverse firewall, are hard to be deployed and experimented with.
- *Reproducible errors and faults* Error identification and isolation is another important task in distributed system evaluation. Current emulation testbeds usually provide error-free environment for testing systems under normal conditions, and have weak or even no efficient mechanisms for the generation and reproduction of software errors or faults.

In this paper, we present vBET, a versatile and scalable emulation testbed based on the virtual machine technology. vBET is formed by one or more physical commodity servers, and therefore readily and locally deployable in any research lab. vBET creates a virtual distributed environment with both network infrastructure and end systems. Each entity, such as a router, a switch, a firewall, or a application-level proxy, is emulated by a virtual machine running unmodified system or application software. Furthermore, the entities emulated by vBET are user-configurable and can be deployed on-demand. The same vBET (physical) server can be easily configured and setup as the testbed for different purposes, such as Internet routing, distributed firewalls, or

¹VLAN technique can alleviate, but can not eliminate re-wiring load

peer-to-peer networks. We will describe the implementation and application of vBET. For the implementation, we present key enabling techniques including virtual OS, virtual topology, small-footprint file system, as well as a network topology modeling language. For the application of vBET, we present the creation of different experiment environments using vBET, including OSPF routing, distributed firewall, and Chord peer-to-peer network. Our experiments demonstrate the versatility, efficiency and scalability of vBET.

The rest of the paper is organized as follows. Section 2 describes the vBET architecture. Section 3 presents a network topology model language, which is powerful to model arbitrary experimental network topology. Section 4 presents the enabling techniques in vBET implementation. Section 5 demonstrates the versatility, configurability and scalability of vBET by setting up environments for a broad range of experiments. Section 6 compares our work with related works. Finally, Section 7 concludes this paper.

2. OVERVIEW OF VBET

Figure 1 illustrates vBET system in operation. Three different experiments are set up in the three vBET servers: vBET server 1 hosts a simple three-node environment for the evaluation of OSPF protocol. vBET server 2 creates a standard multi-LAN environment, and vBET server 3 contains a distributed firewall testing environment. Each *node* inside an emulated environment is a virtual machine, inter-connected by emulated link.

vBET has the following salient features:

- *Easy and local deployment* vBET can be easily and locally deployed in any research lab equipped with commodity servers or high-end desktops. And researchers will enjoy full control over vBET, as well as convenient manipulation and monitoring of their experiments.
- *Flexible and controllable topology deployment* vBET can create a flexible and controllable network topology thanks to the availability of different virtual devices, such as virtual router, virtual switch, and virtual hub. The experimental network topology has no limitation on physical port number available in a router or switch, and no limitation on the number of network connections. Furthermore, in the middle of an experiment, network failures (such as a partition) can be injected, so that researchers can test the reaction of the experimented software to these failures.
- *Customization of core and end system nodes* Although vBET has provided the base implementation for different types of virtual node, every node inside the topology can still be further customized and extended to accommodate new features or it can be entirely replaced with a different implementation.
- *Fast setup of network experiments* Due to the capability of on-demand instantiation and shutdown of virtual machines, vBET achieves highly efficient setup of different experiments. For example, in our sample experiment for the evaluation of *routing flapping in OSPF*, once the virtual topology is specified, it can be set up in about 6 seconds and tore down in about 4 seconds.

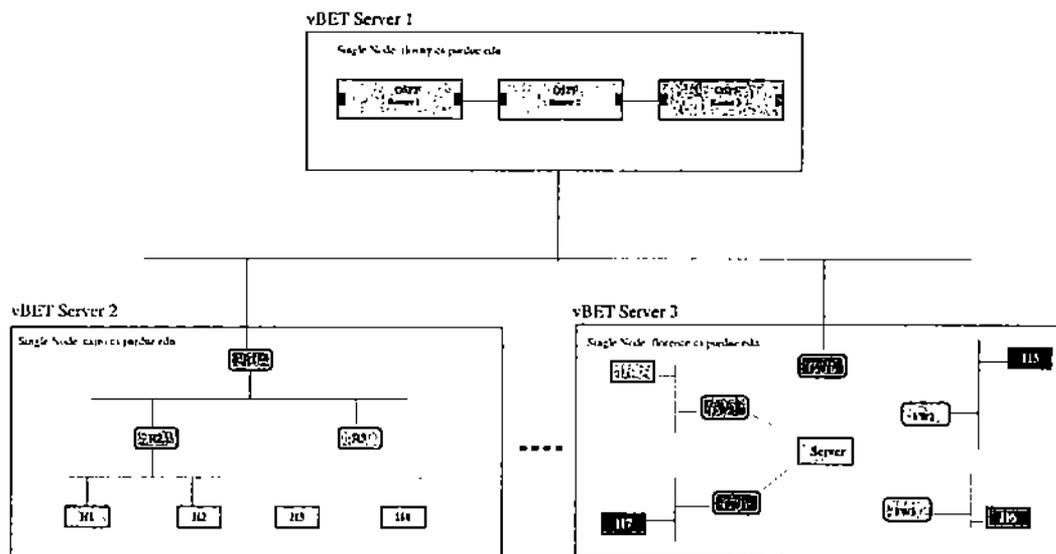


Figure 1: An overview of vBET

- **Scalability** In our vBET implementation, one vBET server (a Dell PowerEdge 2650 server) can emulate up to 60 virtual nodes. This feature can potentially be contributed to current testbeds such as Netbed[26] and PlanetLab[20].
- **Better resource utilization** vBET achieves better utilization than dedicated testbed environment due to the finer granularity of virtual machines and the sharing of vBET server resources among them.

vBET is based on the following key techniques:

- **Virtual OS** Due to the tremendous improvement in server performance, it is feasible to run multiple virtual machines inside one commodity server machine. In our laboratory, one Dell PowerEdge 2650 server with Pentium 4 CPU and 2G memory running linux-2.4.19 enhanced with our CPU proportional sharing capability and customized small footprint root file system, can support up to 60 virtual machines running User Mode Linux (UML) [9], an open-source Linux-based virtual OS. 60 virtual nodes² with arbitrary topology can satisfy the need of a large number of experiments.
- **Resource virtualization** Resource virtualization is helpful for the generalization and allocation of underlying resources to different virtual machines. Resource virtualization techniques, such as *pipe* in [25], can increase the scalability and efficiency of emulation testbeds. *Virtual control terminal* and *virtual network interface card* are two representative examples. Furthermore, wide-area resource virtualization can facilitate wide-area emulations by hiding the specifics of the underlying distributed environment.

²Virtual node and virtual machine are used exchangeably in the following sections

- **Resource sharing** Some existing emulation systems use one physical machine as the minimum unit of allocation, which may result in low resource utilization. PlanetLab[20] defines a *slice* as the allocation unit, so that multiple slices can share the same physical machine and thus achieve better resource utilization. vBET adopts the same philosophy by running multiple virtual machines in the same physical vBET server.
- **Resource and performance isolation** To emulate real-world nodes, virtual machines must have resource guarantees, so that they can run as real nodes with dedicated resources. In other words, the resource and performance of virtual machines needs to be isolated in order to achieve fidelity to the physical nodes.
- **On-demand instantiation and teardown** A virtual machine can be bootstrapped and tore down whenever necessary. Such an on-demand property is desirable for fast experiment environment setup and destruction.

2.1 Phases of Emulation Using vBET

To perform emulation experiments using vBET, there are three main phases:

- **Topology specification** A researcher will use a simple but expressive topology modeling language (to be described in Section 3) to specify the experiment network topology. The topology model language provides several primitives, such as *alloc/dealloc*, *attach/detach* and *link/unlink*.
- **Resource mapping and virtual topology generation** Based on the topology specification, the next step is to map virtual resource requirement to underlying available physical resources. For example, an *OSPF* router should be mapped to a virtual machine which is able to *communicate with* its neighbor *OSPF* routers as specified

in the *OSPF* protocol. During this phase, vBET performs both *virtual node creation* and *virtual topology deployment*. Virtual node must have the specified capabilities, such as running a particular routing protocol or being capable of traffic shaping. On the other hand, virtual topology deployment needs to manage the layout of virtual nodes.

Currently, we have prototyped virtual nodes based on UML but with *extended* functionalities, such as routing, proxying, firewalling and network address translation, so that there is a functionally accurate one-to-one mapping from each node in the topology specification to the corresponding virtual machine. Also virtual switch (or hub) or virtual router (or firewall) can be employed when necessary to glue different virtual nodes. As a result, this phase creates the necessary scripts which are reused in *actual run* phase.

- **Actual run** In the last phase, vBET starts the experiments by invoking the scripts created in previous phases. An experiment may run in a 'slow-motion' mode, due to the constraint of physical vBET server resources. Another key issue is the IP address assignment and network segregation for each virtual node. In vBET system, every virtual node has one unique reserved IP address, and port redirect technique is employed to provide remote researchers with console access to each virtual node for runtime monitoring and management.

3. NETWORK TOPOLOGY MODELING

Modeling network topology helps identify core abstract resource types and their primitive operations. The network topology modeling language of vBET is similar to the facilities provided by ns-2, but easier to understand. Though it is simple, it is powerful for the modeling of network and distributed environments, especially for the composition of complex network topologies based on simple ones.

3.1 Resource Type

Currently, the vBET network topology modeling language supports four different resource types.

- **Network** A *network* represents a medium for communication among network devices. A network can be logical or physical medium depending on the granularity of topological composition.
- **Network Device** The term *network device* is used to refer to the communicating entities, such as *bridge*, *switch*, *router*, *firewall*, *NAT box* or even *end host*, which can generate, forward or accept the real packets and communicates over networks.
- **Network Interface Card** *Network interface card* or *NIC* is the entity which enables the actual packet sending or receiving and is flexible enough to be dynamically attached to or detached from network device.
- **Cable** *Cable* refers to the physical or emulated communication link which enables the real transmission from one network device to another device.

3.2 Primitive Operations

Based on the abstract resource types we have defined, the language further defines three pairs of primitive operations.

- **alloc/dealloc:** A simple resource, such as a *device*, *NIC* or *cable* can be allocated and deallocated for an experiment environment.
- **attach/detach:** One *NIC* can be attached to or detached from a *network device*. For brevity,

$attach(device, NIC_1, NIC_2, \dots, NIC_n)$

can be equivalently used as

$attach(attach(attach(device, NIC_1), NIC_2), \dots, NIC_n)$

$NIC(device, n)$ will return the the n^{th} *NIC* available in the *device*.

- **link/unlink:** *Cable* can be used to link two *NICs* attached to two *network devices*. A link can be broken by the *unlink* operation. The *link* (*unlink*) operation is achieved by performing the action of *plug* (or *unplug*) on both ends of *cable*.

3.3 Examples

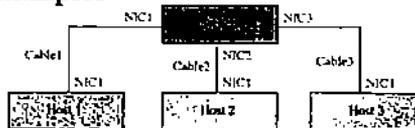


Figure 2: Simple Ethernet topology

To illustrate network topology modeling, we show the following simple example. Figure 2 shows a simple network with three hosts connected by a switch. The detailed modeling script is shown in Figure 3:

```
switch1 = {attach(alloc(switch), alloc(NIC), alloc(NIC), alloc(NIC))}
host1 = attach(alloc(host), alloc(NIC))      host2 = attach(alloc(host), alloc(NIC))
host3 = attach(alloc(host), alloc(NIC))      cable3 = alloc(cable)
cable1 = alloc(cable)                       cable2 = alloc(cable)

plug(cable1, NIC(switch1))  plug(cable2, NIC(switch1))  plug(cable3, NIC(switch1))
plug(cable1, NIC(host1))    plug(cable2, NIC(host2))    plug(cable3, NIC(host3))
```

Figure 3: Network topology modeling script: a simple example

If individual port numbers are self-evident from the context, a simple Ethernet can be simply modeled as shown in Figure 4:

```
Ethernet = link(alloc(switch), alloc(host), alloc(host), alloc(host))
```

Figure 4: A script for Ethernet topology modeling

4. VBET DESIGN AND IMPLEMENTATION

This section presents vBET's design goals of *flexibility*, *scalability*, and *customizability*, as well as its implementation details, including basic building blocks such as *virtual OS*, *virtual networking*, *small footprint file system* and *resource isolation*.

4.1 Design Goals

Besides the versatility for a wide range of network and distributed systems, vBET has the following design goals.

- *Topology flexibility* The network topology can be tailored for one particular distributed system; and multiple network topologies can be created for one experiment environment. Such a flexibility is desirable for on-demand creation of arbitrary network topologies, especially the ones composed by simple network topologies, such as ring, star, or switch-enabled LAN. There should be no physical limitation on the number of physical network connections for each *network device*.
- *Node customizability* Every node in the created network topology can be further customized to experiment with different network services and software (such as different service disciplines and routing algorithms). Current emulation systems have limited or even no support for the customization of core network nodes such as routers and firewalls.
- *Scalability* Instead of scaling the number of physical nodes in the testbed, vBET focuses on the scalability with respect to the number of virtual machines inside one physical vBET server. Current virtual machines techniques, such as VMWare[3] and the original UML[9], are not lightweight enough to enable many physical nodes in a complicated network topology. Our techniques to achieve scalability can potentially be contributed to the existing emulation or real-world testbeds.

4.2 Virtual Machine

To implement virtual machines, there are mainly three levels: *host OS*, *guest OS* and *virtual machine monitor*. Host OS provides the ultimate physical I/O and memory access for guest OS and schedules the guest OS processes as regular processes based on its scheduling policies, such as round-robin or fair queuing [24]. Virtual machine monitor provides fundamental underlying resource virtualization and may be responsible for the accounting of resource consumption³. Guest OS provides a confined environment for all processes running inside it. The isolation includes the *administration isolation*, *fault/attack isolation* and *resource isolation* [16].

vBET supports Linux as the host OS and leverages UML, an open-source virtual OS project. Unlike other virtual machine techniques such as VMWare[3], a UML runs directly in the unmodified *user space* of the host OS; And processes within a UML will be executed in the virtual server exactly the same way as they would be executed in a native Linux machine, which can have performance benefit without the overhead for instruction-level recognition and interpretation such as Java VM [12]. In UML, a special thread is created to intercept the system calls made by all process inside the UML and redirect them into the host OS kernel. Additional process context environment may be created to store or restore upon the entry or exit point of system call, which can reduce the context switch overhead, and thus increase the scalability and availability of VM.

³Host OS can also assume the responsibility of resource accounting if one unique ID can identify a virtual machine.

We have extended UML virtual machine, especially from host OS perspective to enable proportional resource sharing, and implemented small footprint root file system for UML, which significantly increases the scalability of UML.

4.3 Virtual Networking

Virtual networking enables the communication among virtual nodes, and is essential to topology flexibility. We classify different virtual nodes according to their roles as follows:

- *Virtual hub/switch* Like a regular physical hub (switch), virtual hub (switch) enables simple packet forwarding to construct one simple LAN environment. Virtual hub will forward every packet received to every available port, which may result in degraded performance due to the multiple copies of packets. Virtual switch adds intelligence to packet forwarding, so that only designated receivers will receive the packet. In the vBET prototype, one temporary UNIX domain socket is created as the concentration point for one particular virtual hub (or switch), and it can receive incoming connection request and create one additional virtual port for incoming connection, which basically eliminate the physical port number limitation. Advanced packet queuing and forwarding policies can be applied to virtual hub (or switch) to model any link characteristics, such as link bandwidth, latency, loss rates and congestion.
- *Virtual router/firewall* A router or firewall operates as one part of the network infrastructure and enables communication between *network devices* which are not connected to a common network. Router mainly focuses on the fast packet forwarding and firewall performs the packet filtering and content inspection. We have designed a multi-purpose minimal *ext2* file system, which is extensible for different functionality modules (Section 4.4). Different functionality module can be included during the *resource mapping and virtual topology generation* phase, or more specifically *virtual node generation*.
- *Virtual link/NIC* Virtual link is constructed or deconstructed during the connection establishment or teardown with one particular virtual switch (or hub). We shift modeling responsibility of link characteristics to virtual hub/switch for easy prototyping. Virtual NIC is made available as TUN/TAP device or virtualized as another UNIX domain socket who make the connection request to the virtual switch (or hub). As mentioned before, there are no practical limitation on the number of available virtual NICs for each network device.

4.4 Root File System

Upon finishing the boot sequence, the Linux VM kernel attempts to locate and mount a root file system. In order to increase the scalability of VMs, the root file system need to have small memory footprint, and should be tailored for a specific purpose by excluding unnecessary services. In our prototype, we impose the space size (32 MBytes) for holding root file system (with type *ext2*), which is reasonably

small but contains sufficient basic functionalities. Also the base root file system should be extensible to include optional packages, which are associated with a general domain or application.

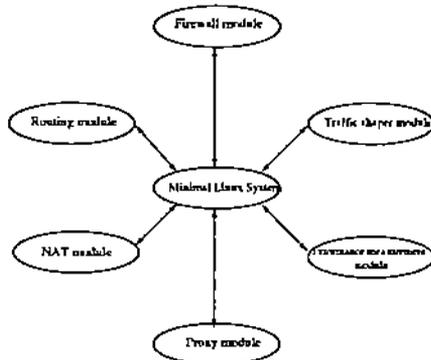


Figure 5: Optional file system packages and the minimal Linux system

Figure 5 illustrates the set of six optional packages. The *Firewall module* contains code that implements an Internet firewall subsystem, and it provides stateless or stateful packet filtering. The *NAT module* handles packet header translation and even packet payload mangling (such as for the FTP data connection) needed for network address translation. NAT can be integrated with a firewall or be used separately. The *routing module* contains code for routing protocols, including RIP, OSPF, and BGP taken from *zebra* [15] package. The *Proxy module* provides functionality of SOCKS proxy services, and *Traffic shaper module* can be employed to experiment with transmission rate monitoring and policing. *Performance measurement module* includes some performance measurement routines, such as *tcp* [2]. All of the modules depend on the underlying *Minimal Linux system*, which contains basic system⁴-wide configuration, daily routines and is thus required for operation of each virtual node.

4.5 Resource Isolation

To achieve performance isolation of each virtual node, resource sharing and isolation need to be guaranteed by underlying host OS or virtual machine monitor. Resource isolation can prevent malfunctioning virtual node from monopolizing the CPU or *eating up* all of available network bandwidth. Resource isolation is necessary for creating high-fidelity emulation environment. Via extensions to host OS (Linux), vBET implementation supports CPU, network bandwidth, and memory isolation.

- *CPU capacity isolation* is achieved by implementing a coarse-grain CPU proportional sharing scheduler in the Linux host OS. The scheduler enforces the CPU share allocated to each virtual node. The CPU sharing of a virtual server is decided during the *virtual node generation*. Usually every virtual node gains the fair share of CPU cycle if not explicitly specified. Within one virtual node, all processes bear the same user id,

⁴Here, *system* means one particular virtual node

and host OS CPU scheduler performs resource consumption accounting and enforces resource usage limitation for virtual nodes based on their user id.

- *Network bandwidth isolation* includes the *internal traffic isolation* and *external traffic isolation*. Internal traffic is restricted inside the virtual network topology, and will not consume real-world network bandwidth. External traffic generates real network traffic between vBET servers. For internal traffic, we equip each upstream virtual node with *traffic shaping capability* by including *traffic shaper module*. For external traffic, a traffic shaper running inside the host OS enforces the fair share of outbound bandwidth among virtual nodes. Since every virtual node has its own IP address, the traffic shaper can easily identify and thus charge the node responsible for each outgoing packet.
- *Memory isolation* is critical for the performance of virtual nodes. vBET simply leverages the *memory usage limit* feature of UML: the maximum amount of memory available to a virtual node can be specified when the virtual node is started.

5. CASE STUDY

We have created several interesting experiment environments to demonstrate the salient features of vBET: The first experiment tests *routing flapping in OSPF*, which shows an infrastructure-critical routing protocol can be deployed and customized in the core nodes. The second experiment is in another interesting application domain: *distributed firewall*. One flexible network topology is created to enable the coordination of a set of distributed firewalls to protect one central server. The third experiment emulates application-level peer-to-peer lookup service, and subjects the service to different failure models (such as network partition) under different network topologies.

5.1 Routing Flapping in OSPF

This experiment examines an interesting scenario involving the OSPF protocol, which demonstrates that baneful persistent route flaps may exist. The logical network setup for this experiment is shown in Figure 6. The corresponding topology modeling script is shown in Figure 7. To highlight the efficiency of vBET, we note that the whole system is bootstrapped within 6 seconds, and can be torn down within 4 seconds.

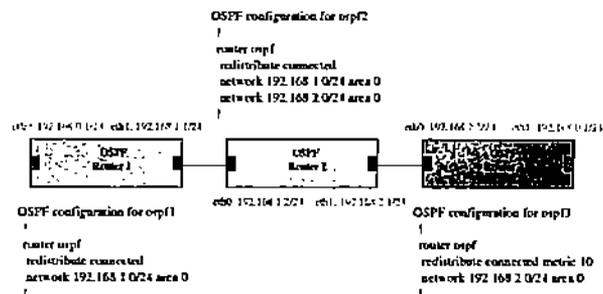


Figure 6: Network topology for testing routing flapping in OSPF

In Figure 6, there are three routers - R_1 , R_2 and R_3 , each of which is running ospfd from Zebra [15] routing software. Each of routers R_1 and R_3 intentionally has one interface configured to have the same IP address 192.168.0.1/24, and router R_1 advertises route entry for 192.168.0.1/32 with metric 20 (default value) and router R_3 advertises same destination with smaller metric 10. Since router R_3 incurs lower cost to reach destination 192.168.0.1/24, router R_2 should choose R_3 for destination 192.168.0.1/24 when the network is stabilized. Detailed OSPF configuration and IP address assignment for each router are shown in Figure 6.

```
r1 = attach(alloc(router), alloc(NIC), alloc(NIC))
r2 = attach(alloc(router), alloc(NIC), alloc(NIC))
r3 = attach(alloc(router), alloc(NIC), alloc(NIC))

link(NIC(r1,2), NIC(r2, 1))
link(NIC(r2,2), NIC(r3, 1))
```

Figure 7: Network topology modeling script for the testing of routing flapping in OSPF

At the beginning, router R_1 and R_2 are bootstrapped, then R_3 is started. When routing tables are stabilized in all three routers, from the screenshot in Figure 8, we can see the router R_2 (i.e., ospf2 in Figure 8) adopts the route to 192.168.0.1/24 via router R_3 (i.e., ospf3 in Figure 8), which is correct since R_3 contains smaller metric for destination 192.168.0.1/24.

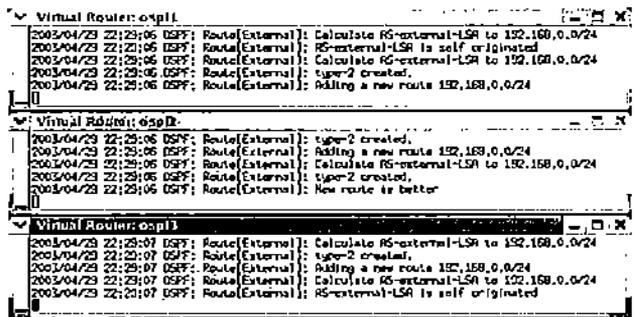


Figure 8: Screenshot of OSPF experiment when all routers are working

When we intentionally disable interface 192.168.0.1/24 in router R_3 , R_3 notifies R_2 that the corresponding link state age has reached *MaxAge*, and thus is considered not usable anymore. As a result, a more expensive route (with metric 20) to 192.168.0.1/24 via router R_1 (ospf1 in Figure 9) is adopted by router R_2 .

Then, we re-enable interface 192.168.0.1/24 at router R_3 to examine the routing flapping effect in OSPF. Figure 10 shows that router R_2 has found the new route via router R_3 is better than current route via router R_1 for destination 192.168.0.1/24. As a result, better route for the destination is updated in router R_2 .

After several rounds of enabling and disabling, persistent route flapping effect is clearly exhibited. Route flapping is a baneful phenomenon and needs to be eliminated for more stable and robust networks. Also it infers that more ad-

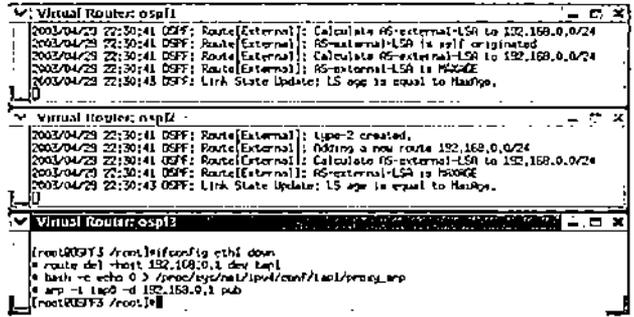


Figure 9: Screenshot of OSPF experiment when the better route to 192.168.0.1/24 is down

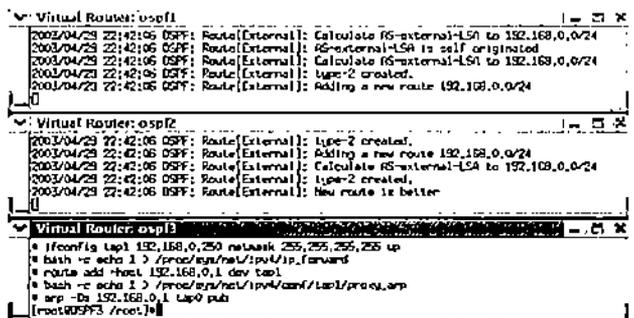


Figure 10: Screenshot of OSPF experiment when better route to 192.168.0.1/24 is up again

vanced OSPF route flap damping features should be introduced and the OSPF protocol performance can be improved by learning from the routing flapping history and be lazy and conservative in adopting new or better route entries. To the best of our knowledge, this is the first time OSPF route flapping effect can be so vividly demonstrated by an emulation testbed. Our trace logs from this experiment are available at <http://www.cs.purdue.edu/homes/jiangx/vBET>.

5.2 Distributed Firewall

This experiment enables the creation of a topology shown in Figure 11, in which several Internet connections are available to provide web services. A set of *edge firewalls* examining the incoming requests for the service need to be coordinated to ensure that the total request load for the service does not exceed the server capability. The whole system is bootstrapped within 13 seconds, and can be torn down within 7 seconds.

The virtual network topology is modeled by the script in Figure 12.

We create and compare two simple scenarios to demonstrate the effectiveness of distributed firewall. In the first scenario, every firewall forwards requests to the central server without any limitation, similar to a DDoS attack on central server. In the second scenario, every firewall restricts the traffic toward the central server in order to prevent the DDoS attack pro-actively. Figure 13 captures the screenshot of an *actual run*.

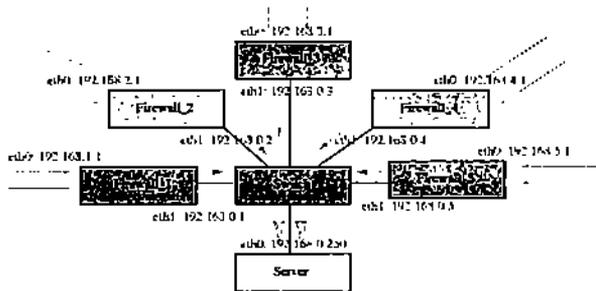


Figure 11: Distributed firewall testing environment

```

server = attach(eth0 to) eth0(NIC)
fw1 = attach(eth0 to) eth0(NIC)
fw2 = attach(eth0 to) eth0(NIC)
fw3 = attach(eth0 to) eth0(NIC)
fw4 = attach(eth0 to) eth0(NIC)
fw5 = attach(eth0 to) eth0(NIC)

eth0(eth0(=eth0, server, NIC(=1), NIC(=2), NIC(=3), NIC(=4), NIC(=5), 1)

```

Figure 12: Network topology modeling script for the distributed firewall environment

We measure the traffic rate observed by the server in both scenarios. In the first scenario, the server can receive traffic up to 84.66Mbps. In the second scenario, traffic destined to the server is limited to 640kbps at each firewall with `tc` command and as a result, the server only receives traffic at a rate of 2.8Mbps. The individual traffic rates via the five firewalls are shown in Table 1.

Though conceptually simple and straightforward, such experiment is difficult, if not impossible, to setup and experiment with in real world without a dedicated testbed. PlanetLab is not helpful for this purpose since the nodes are deployed as end systems. Netbed is able to experiment with small scale distributed firewall, but has limitation on the network topology and on the number of network connections available to routers in the topology. In addition, it is difficult to isolate the traffic of this experiment from traffic generated by other experiments.

5.3 P2P Network

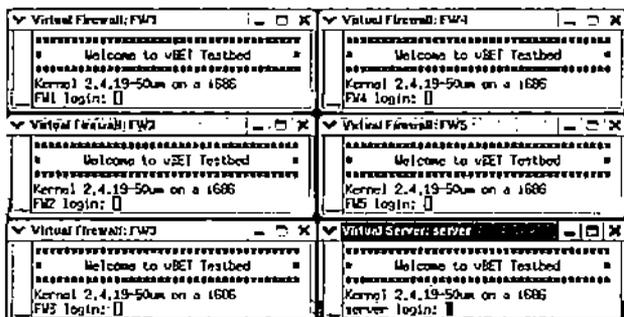


Figure 13: Screenshot of distributed firewall experiment

Firewall	Amount of traffic
FW1	578.8kbps
FW2	579.9kbps
FW3	579.4kbps
FW4	579.9kbps
FW5	580.0kbps

Table 1: Inbound data (request) rate regulated by each firewall

Finally, we create Chord, a peer-to-peer overlay lookup service in vBET. The peer-to-peer network is deployed over nodes in a network topology depicted in Figure 14. The entire network is bootstrapped within 2 minutes, and can be torn down within 1 minute.

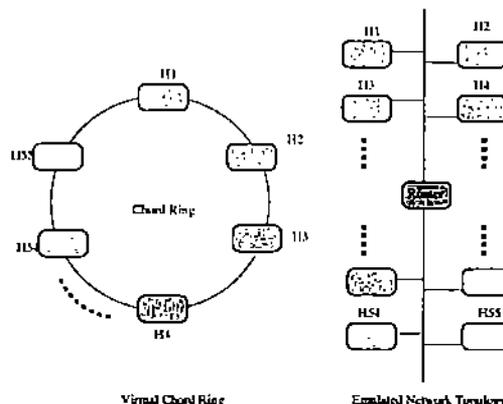


Figure 14: A Chord P2P overlay network with 55 peers

In figure 14, there are 55 Chord peers, which are instantiated around the Chord ring (see screenshot in Figure 15). One Chord node assumes the responsibility as bootstrapping node. The 55 chord nodes resides in two LAN networks, which is connected by a virtual router. Such topology is useful to demonstrate and validate the reaction of Chord to different failure models, especially the network partition failure.

More complicated topologies can be created to experiment different aspects of Chord network, such as CFS storage utility [8], the peer-to-peer storage service based on Chord. Our purpose here is to show vBET's flexibility of creating experimental network topology and its customizability for every node inside the topology. One of our on-going projects is to design and apply trust models and reputation management systems to overlay networks, and test them using vBET.

6. RELATED WORK

There have been some previous efforts in investigating the use of emulation in the context of their specific research [13, 19, 14]. Recently, several general-purpose emulation testbeds have been proposed and deployed [20, 26, 25, 4]. vBET shares the same goal of providing high-fidelity emulation or real environment for the experimentation with distributed systems and network protocols.

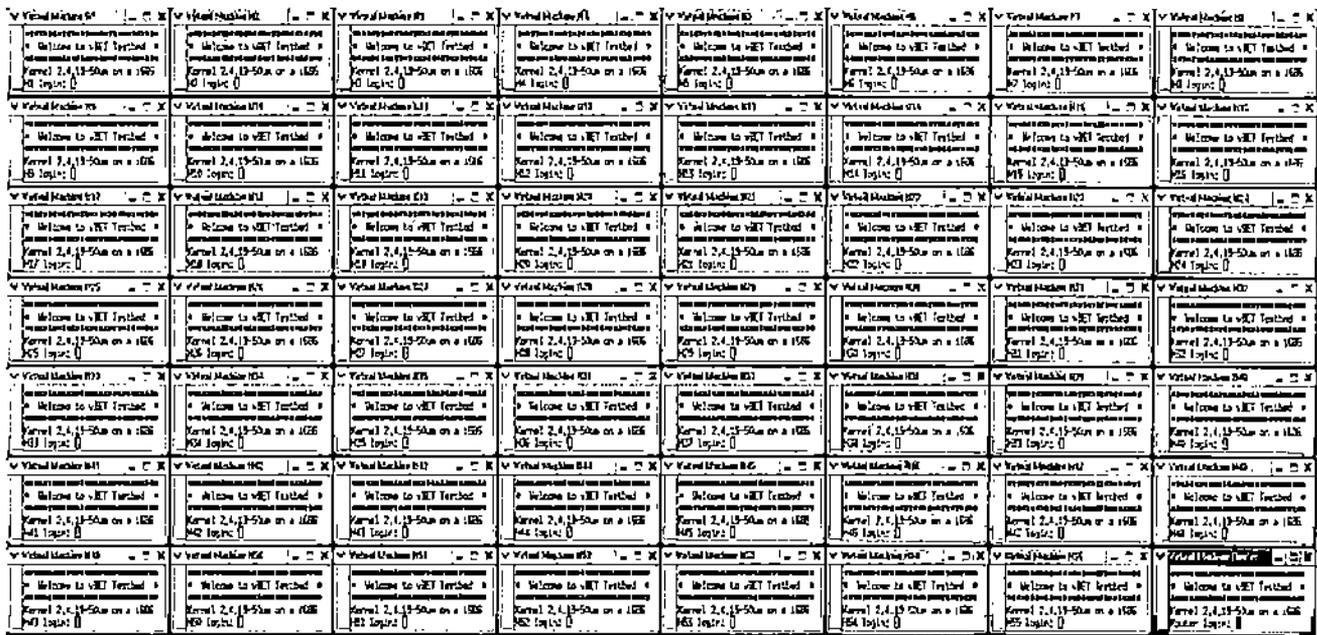


Figure 15: Screenshot of Chord P2P network experiment

PlanetLab [20] is in the process of deploying hundreds of nodes across the Internet to create Point of Presence (or PoP), so that wide-area distributed systems can be deployed and evaluated using real Internet traffic. Though extremely valuable because of real deployment, the experiment environment is difficult for an individual researcher to gain full control. And core routers are not allowed to run customized software. As a result, it may not be feasible to experiment with infrastructure-critical systems and protocols, such as distributed routing and distributed firewall. vBET complements PlanetLab: the former can be installed in a PoP of the latter, so that experiments with infrastructure-critical systems can be carried out locally.

Netbed [26] allows users to configure a subset of network resources for isolated distributed systems and networking experiments. It provides an integrated environment that allows users to set up target operating systems and network configurations. Again, vBET can be integrated with Netbed, especially by contributing the scalability (with respect to the number of virtual nodes within one physical machine) and flexibility (of topology setup) features to the latter.

ModelNet [25] targets scalable and flexible emulation environment, which can perform full hop-by-hop network emulation. The basic mechanism for scalability is the pipe technique from *dummyNet* [22]. As a result, it is difficult for ModelNet to support *traceroute*-like applications. One important property of ModelNet is that it balances scalability and efficiency by employing both emulation and simulation techniques. vBET, on the other hand, provides complete control and customization over core nodes. It can run third-party softwares in core nodes, including routing protocol, advanced stateful packet filtering or other network services.

Umlsim[4] is another recent effort parallel to ours which exploits the virtual machine technology to create simulation/emulation platforms. Instead of focusing on a specific application or protocol (such as TCP), vBET aims at providing a general-purpose testbed for different applications and protocols.

7. CONCLUSION

Emulation testbeds are expected to be easily and widely deployable. In an emulation testbed, arbitrary network topology can be created for different experimental systems and protocols. Furthermore, it is desirable to support core node customization, including fault injection into core nodes, coordination between core nodes, as well as update or modification of critical components in core nodes, such as routing protocol. In this paper, we present the design and implementation of vBET, as our initial efforts towards meeting these goals. The salient features of vBET include easy deployment, scalability, customizability, and efficiency. Our experiments with different network and distributed systems demonstrate the versatility of vBET. Furthermore, vBET complements existing emulation or real-world testbeds and can be readily integrated into the existing systems.

8. REFERENCES

- [1] The Network Simulator ns-2. <http://www.isi.edu/nsnam/ns/>.
- [2] *ttcp*. <ftp://ftp1.sunet.se/pub/network/monitoring/ttcp>.
- [3] VMWare. <http://www.vmware.com>.
- [4] W. Almesberger. UML simulator. <http://www.almesberger.net/umlsim/>.
- [5] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. *Proc. 18th ACM SOSP, Banff, Canada*, Oct. 2001.
- [6] R. Braynard, D. Kostic, A. Rodriguez, J. Chase, and A. Vahdat. Opus: an Overlay Peer Utility Service. *Proceedings of the 5th International Conference on Open*

- Architectures and Network Programming (OPENARCH)*, June 2002.
- [7] A. Chervenak, I. Foster, C. S. C. Kesselman, and S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets. *Proceedings NetStore'99*, Oct. 1999.
- [8] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, Oct. 2001.
- [9] J. Dike. User Mode Linux. <http://user-mode-linux.sourceforge.net>.
- [10] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. *HotOS VIII, Schloss Elmau, Germany*, May 2001.
- [11] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum*, June 2002.
- [12] E. Gagnon and L. Hendren. SableVM: A Research Framework for the Efficient Execution of Java Bytecode. *Java Virtual Machine Research and Technology Symposium (JVM '01)*, Apr. 2001.
- [13] G. Banga, J. Mogul, and P. Druschel. A scalable and explicit event delivery mechanism for UNIX. *Proceedings of the USENIX 1999 Annual Technical Conference, Monterey, CA*, June 1999.
- [14] H. Yu and A. Vahdat. The Costs and Limits of Availability for Replicated Services. *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2001.
- [15] K. Ishiguro. Zebra. <http://www.zebra.org/>.
- [16] X. Jiang and D. Xu. SODA: a Service-On-Demand Architecture for Application Service Hosting Utility Platforms. *Proceedings of The 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12), Seattle, WA*, June 2003.
- [17] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, Nov. 2000.
- [18] J. Moy. OSPF Version 2. <http://www.ietf.org/rfc/rfc2328.txt>, Apr. 1998.
- [19] B. Noble, M. Satyanarayanan, G. Nguyen, and R. Katz. Trace-Based Mobile Network Emulation. *Proceedings of ACM SIGCOMM 1997, Cannes, France*, Sept. 1997.
- [20] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *Proceedings of ACM HotNets-I Workshop, Princeton, New Jersey, USA*, Oct. 2002.
- [21] Y. Rekhter and T. Li. Border Gateway Protocol 4 (BGP-4). <http://www.ietf.org/rfc/rfc1771.txt>, Mar. 1995.
- [22] L. Rizzo. Dumynet and Forward Error Correction. *Proceedings of the USENIX Annual Technical Conference*, June 1998.
- [23] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. *Proceedings of SIGCOMM'98*, Sept. 1998.
- [24] V. Sundaram, A. Chandra, P. Goyal, and P. Shenoy. Application Performance in the QLinux Multimedia Operating System. *Proceedings of the Eighth ACM Conference on Multimedia*, Nov. 2000.
- [25] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.
- [26] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.