Australasian Journal of Educational Technology

2005, 21(2), 263-282

Teaching IP networking fundamentals in resource constrained educational environments

Grenville Armitage and Warren Harrop
Swinburne University of Technology

Many educational institutions suffer from a lack of funding to keep telecomm-unications laboratory classes up to date and flexible. This paper describes our Remote Unix Lab Environment (RULE), a solution for exposing students to the latest Internet based telecommunications software tools in a Unix like environment. RULE leverages existing PC laboratories (often based on Microsoft's Windows) to enable student access to Internet Protocol (IP) networked hosts for telecommunications coursework and research projects. Re-use of existing PC labs substantially decreases the cost of introducing hands on teaching of Unix based Internet services into curricula. We discuss our experiences of deploying, using and provisioning RULE since early 2003. RULE itself is a handful of FreeBSD hosts, mounted in a small back room, utilising FreeBSD's "jail" functionality to create multiple virtual hosts.

1. Introduction

Our Telecommunications and Networking group faces a challenge common to many tertiary educational institutions – how to keep our students abreast of the latest IP (Internet Protocol) networking services and technologies, while working with scarce laboratory space and limited funding. We had a well developed and extensive set of PC Labs scattered around campus, predominantly running Microsoft's Windows. However, we now wanted our students to 'get their hands dirty' by actually installing and using IP based server and client applications from the open source community [1], for example, web servers like Apache [2], web crawlers/indexers and web proxies, alternative file servers such as Samba [3], or even running their own Domain Name System servers. Our students would learn how to use, modify and rebuild these applications.

We had also decided in late 2002 that a strong emphasis on Windows PC environments needed to be balanced by an increased exposure to Unix like

environments, particularly in the context of IP networked services. The school's existing PC labs were frequently booked solid for classes run by a variety of departments and we also wanted to avoid the additional cost (in time and salary) of recreating/rebooting machines just for our IP networking classes. Building a separate computer lab, with dedicated machines, desk space and infrastructure support, was considered an expensive last resort.

Our solution is the Remote Unix Lab Environment (RULE) (Figure 1). RULE provides multiple networked Unix like hosts without requiring additional dedicated lab space for Unix only machines. The existing campus PC labs are used as terminals through which students access their assigned RULE hosts. Because access is via our campus network, students can also engage in project work from home or from wireless equipped laptops. RULE itself is housed in a standard 19 inch rack and tucked away in a corner of a small room, meeting our goal of minimal additional space and infrastructure costs.

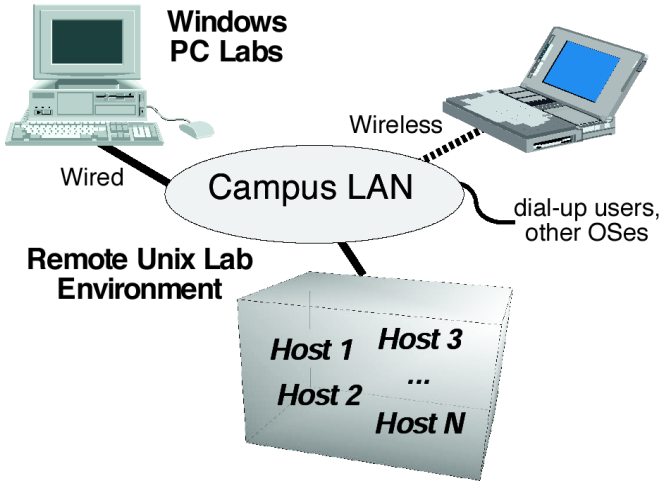

Figure 1: Remote Unix Lab Environment is accessible from networked machines around the campus

RULE is built around FreeBSD [4], a robust, well supported and free Unix like operating system that runs on a range of common and inexpensive PC motherboards. FreeBSD can instantiate multiple virtual hosts on a single motherboard, multiplying the number of students we can support with a limited set of physical hardware.

Although our teaching staff are technology literate, they are not trained as system administrators. Thus, we developed a Jail Host Toolkit (JHT) to create and manage RULE virtual hosts (so named because the virtual hosts are enabled by FreeBSD's 'jail' functionality [5]). JHT started as a set of scripts and has evolved into a single application interface for staff to manage RULE hosts in an effective and clean fashion. Our first generation of RULE was based on FreeBSD 4.7. We later moved to FreeBSD 4.9 and are currently running RULE and JHT under FreeBSD 5.3.

In section 2 we outline the various approaches we could have used to instantiate RULE, and section 3 elaborates on our reasons for choosing FreeBSD jail hosts. In section 4 we describe our experiences using RULE – types of classes, student experiences and our experiences as teachers and lab demonstrators. Section 5 discusses how campus network security is affected by RULE, while section 6 provides a relatively detailed analysis of how well RULE's jail hosts function on different PC motherboard configurations. Section 7 wraps up with a brief review of JHT, the open issues with respect to resource management between jail hosts, and our future plans for RULE.

2. Possible solutions

There are a number of technical approaches we could have pursued to instantiate a RULE system meeting our pedagogical and cost goals. These goals were:

- Provide students with self directed access to internet applications (eg. clients, servers, and / or proxies) that they can compile, install, trial, modify and rebuild / reconfigure with minimal supervision.
- Allow students to access RULE from anywhere on the campus intranet.
- Protect the rest of the university (and wider Internet) from student activities inside the RULE.
- Utilise off the shelf components wherever possible and work in a 'headless' configuration (no directly attached monitors, keyboards or mice).

First we considered our target operating system environment. We realised that Unix like environments typically provide the most flexible contexts within which to deploy, use and reconfigure open source IP networking software. In principle this could be achieved in a number of ways, including Microsoft Windows installations combined with Cygwin [6] (an open source, Unix like application layer environment for Windows) or free Unix like offerings such as Linux [7], FreeBSD, OpenBSD [8], and NetBSD [9].

Second we considered the physical realisation of our RULE system – a central multi-user machine, multiple dedicated machines, multiple virtual machines or multiple virtual hosts. A central multi-user machine (accessed from remote terminals, which may themselves be PCs in standard PC labs, fails to provide our target learning environment. Each student needs to feel like they 'own' the IP host on which they're doing their experiments, yet only one user account at a time can run a network client or server on a particular 'port' numbers. For example, only one user account would be able to run a webserver on the traditional 'port 80', with other students forced to use alternative ports, diminishing the learning experience.

The most flexible, albeit expensive, approach is to implement a laboratory of individual desktop computers - either entirely dedicated to the task [10] or dual booting some combination of operating systems. Some institutions have gone as far as to fully support dual boot Windows/Linux on entire lab setups on a wide scale [11]. A closely related variation is to 're-image' these lab machines (effectively re-installing the operating system from scratch) at the beginning of each lab class, a time consuming option and something we were trying to avoid.

Virtual machines and virtual hosts are an attractive alternative to dedicated machines when space and cost are at a premium. In both cases a single physical computer emulates many machines or hosts. The virtual machines or hosts would be accessed remotely by client applications running on existing campus lab machines (which do not, themselves, need to be running a Unix like operating system). Tools like ssh [12] VNC (remote desktop viewing software) and X11 (a network GUI allowing applications to display on remote machines) can all be used for remote access.

Virtual machines, such as those implemented by VMware [13], Bochs [14], Plex86 [15] or QEMU [16], provide virtual hardware environments within which a number of different operating systems may be concurrently operating. Running multiple kernels under VMware has been described recently to be an effective method of teaching operating system development [17], because the students can rebuild entirely custom kernel environments and have full control over the virtual machine.

Virtual machines may fully emulate the desired hardware environment in software, or employ a mixture of emulated and 'real' execution contexts. In the former case the processor being emulated may be entirely different to the real hardware on which the virtual machines are being run. In the latter case, the virtual machines have the same architecture as the real underlying hardware. At the time we began development of RULE, virtual machines were either very slow (eg. orders of magnitude slower than the underlying

host hardware in the case of full emulation with Bochs) or expensive (such as the effective, yet commercial, VMware).

Virtual hosts are a less complex alternative to virtual machines. Rather than virtualise an entire hardware environment, virtual hosts create concurrent, distinct user space 'host' environments on top of a single instance of a running operating system kernel. This single kernel mediates access to shared resources such as disk, network, and general I/O ports. By 2002 a number of Internet Service Providers (ISPs) were using FreeBSD's in built 'jail' functionality to create virtual webhosting services. Since virtual hosts are entirely sufficient to meet our stated goal of exposing students to typical Internet applications, we decided to explore the use of FreeBSD as our target operating system environment.

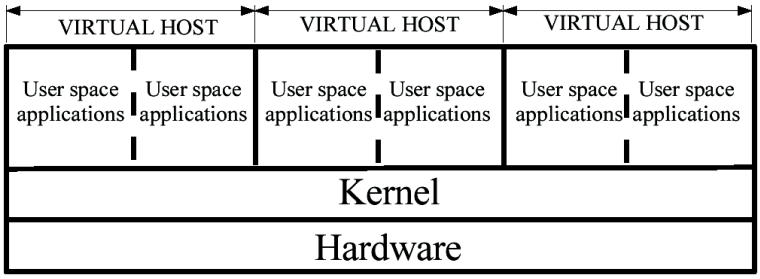

Figure 2: Virtual hosts share a common kernel

3. Choosing FreeBSD for virtual hosts

Ultimately there were three key reasons why we chose to base RULE around FreeBSD:

- FreeBSD's clean 'packages' and 'ports' mechanisms (for installing applications in pre-compiled and compile as needed forms) provides students with a number of ways to experiment with thousands of pre-existing networked applications.
- Many application binaries compiled under Linux also run directly under FreeBSD and most of these can be recompiled as native FreeBSD applications if needed.
- FreeBSD has in built and well understood facilities for creating virtual hosts.

Virtual FreeBSD hosts are a central part of RULE. We developed a Jail Host Toolkit (JHT) to simplify the establishment and management of multiple virtual hosts on a single physical motherboard. Because JHT virtual hosts

are implemented using the FreeBSD kernel's `jail(8)` functionality, we refer to them as jail hosts. The machine in which jail hosts reside is referred to as the primary host.

Jail hosts are replicas of the FreeBSD user space environment, each with their own distinct IP addresses, user accounts and the ability to run separate instances of most user space applications. Once a jail host has been configured and booted, it appears just like a regular, IP accessible FreeBSD machine. For example, Figure 3 captures the equivalence between three independent hosts on an IP network and a single FreeBSD machine supporting three jail hosts – from elsewhere on the network, these two scenarios are largely indistinguishable.

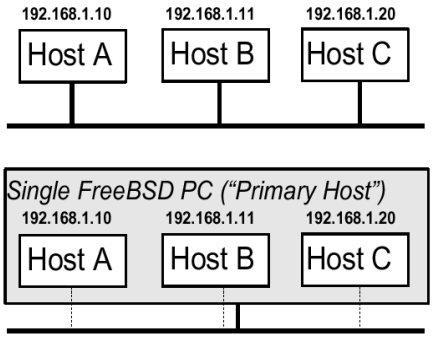

Figure 3: Jail hosts appear as independent IP hosts on the network

In essence RULE is a collection of jail hosts that provide many more independent FreeBSD environments than we have physical motherboards, thereby saving valuable floor and rack space. Using FreeBSD jails ultimately came down to the questions of performance, price and flexibility. Our initial functional goals for RULE could have been met by running multiple virtual machines on a single primary host.

However, the overhead of emulating multiple concurrent kernel and user space environments seemed totally unnecessary, particularly given our focus on student access to user space network applications.

If our focus had been on kernel development and experiments with kernel resident networking code then virtual machines would have been a compelling choice. However, when our educational goals evolve to this level we will also investigate simply deploying multiple, independent small form factor motherboards for each student's use.

4. Experiences with RULE

Our experiences with rule can be roughly broken into three categories. Examples of what we have been teaching with RULE, the teaching experiences and the student's experiences. We will briefly touch on all three of these topics.

4.a. Lab class curriculum

RULE first saw operational use in March 2003, and has since been used successfully over three semesters and five different semester long classes. Our students remotely access their RULE hosts (one or more each, depending on the particulars of each lab class) from regular Windows based desktop PCs.

Our typical lab classes involve students accessing their jail hosts using command line and remote file copying tools based on 'secure shell' (ssh) [12], giving them a multiplatform client/server environment in which to work. From Windows based lab machines we use the free software packages PuTTY [18] and WinSCP [19], while students accessing RULE from Linux or FreeBSD machines typically have their own equivalent open source tools.

Students do all their work within jail hosts. They can be given hands on experience of having root (administrative user) access within their jail host, (managing group and user accounts, access levels, system services, etc) without compromising other jail hosts or the primary host. Students can run their applications on “well known” port numbers without conflicting with similar applications being run by other students whose jail hosts share the same primary host. (Neither of these would be possible if we placed all the students on a single, central multi-user host and differentiated them solely by login name or user ID.) RULE was first used with a class of final year undergraduate Telecommunications Engineers. We tasked them with writing and testing their own web proxy using one of the three programming languages C, C++ or Java. In addition to their regular lab Windows PC (with a standard university software configuration) each student effectively had their own dedicated Unix machine on which to develop and debug their web proxy. Students were able to initiate web page requests from their Windows desktop (either from Internet Explorer or Netscape) and see their desktop client connecting to, and eventually communicating through, their own web proxy.

A postgraduate (Masters) class in 2003 made use of RULE to experiment with installing, configuring, and modifying the open source Apache web server. As with the undergraduates, RULE allowed students to launch

client requests from their standard Windows/Internet Explorer desktop and see their desktop client(s) accessing their own custom configured web server. Being able to control both the client and server side of an http (web) exchange enabled the students to make a much clearer association between their configuration actions and the consequent effect on web service delivery.

In the first half of 2004 we utilised RULE on a larger scale (60+ jail hosts rather than the 20-25 jail hosts deployed in 2003). Qualitative feedback from students is that RULE host performance is sufficient for their class work (we discuss RULE performance further in section 6).

RULE allowed our students to become familiar with Unix like command line environments, to setup and configure a small http server (a version known as 'thttpd', or 'tiny httpd' [20]), setup and configure a web proxy (a version known as 'tinyproxy' [21]), and setup and configure their own Domain Name Service (DNS) server [22]. Two other classes (part of our postgraduate coursework program) have also been leveraging RULE hosts to provide platforms for student-configured web servers and back-end database operations (e.g. using the PHP scripting language [23] and mySQL database package [24] under FreeBSD).
In each case:

- Students gained valuable experience in a multiplatform environment, seeing first hand how Unix and Windows systems interoperate over IP networks.
- Students saw their networked services respond to their configuration changes.
- We leveraged the university's existing PC labs as the student's “front end” to RULE.

RULE also enables class experiments involving content “server farms” on the in house network. With a single primary host platform a student can emulate a cluster of independent web, ftp or email servers. Nevertheless, jail hosts do impose certain limitations on what networking experiments students can perform. Even if logged in as administrator (root) within a jail host:

- Students cannot run programs that require raw access to the underlying IP or Ethernet layers. This precludes the use of common network probing commands such as ping and traceroute.
- Students cannot access or modify the routing tables of the underlying primary host.
- Students cannot rebuild the kernel, or modify the running kernel's state.

These restrictions are generally acceptable when teaching students about client/server applications. More advanced 'low level' networking classes, focused on Ethernet traffic monitoring, routing and switching, will need a version of RULE that allows direct access to primary hosts. Appendix A briefly discusses the restrictions associated with jail hosts in more technical detail.

4.b. Teaching experiences

Having high level control of every student's RULE hosts from the primary host level provides lab demonstrators with a number of benefits. JHT allows lab demonstrators to remotely 'log into' any student's RULE host as an administrator, without knowing any of the student's passwords. This allows quick debugging of student inquiries about their work. Teachers can investigate problems from within the context of the student's work in progress, issue commands to locate (and possibly correct) the student's problem and provide necessary feedback to the student. This process has proven invaluable when solving after hours inquiries from students (usually via email).

A number of teaching and administrative benefits accrue from the fact every RULE host's 'hard drive' is in fact a regular directory on the primary host.

Teaching staff logged into the primary host can read/write any file on a student's RULE host, which we have found especially helpful. Students sometimes 'mangle' various configuration files beyond the point where they can be successfully read. In these situations the student can be taken back an arbitrary number of steps in their lab work by having their 'broken' files replaced by known good copies residing on the primary host. The student can then continue from an earlier point in the lab session knowing things are in working order. Administratively it is trivial to run regular file system backups of each jail host from the primary host level, without requiring any action by the student in charge of each jail host. An entire jail host – user files, system files, etc - can be archived and restored with tools as simple as tar or rsync [25]. Indeed, the administrator (root) account on each jail host *cannot prevent* such backups from being performed. This has at least three interesting consequences:

- Students can be protected from themselves - if they accidentally delete an important part of their jail host's file system, it can be restored from the regular backups.
- File system snapshots at the primary host level can track the evolution of each student's work - a useful audit trail in cases where you suspect one student (or lab group) has augmented their work with solutions from another student.

- Automated tasks on a primary host can switch around the identities of jail hosts according to class schedules - the equivalent of re-imaging lab machines from CD, but without the labor intensive intervention of a staff member. (There can be many more jail host file systems stored on the primary host than there are active jail hosts at any one time.)

From the primary host, it is also quite simple to monitor student login and logout events recorded by each jail host, file system consumption within each jail, and generally obtain a globally scoped perspective on what is going on across the jail hosts. This makes class supervision and control far simpler than if students had their own, independent Unix hosts each on its own motherboard. For example, during the lab class run in early 2004 and early 2005 we collected lab reports from 'well known' home directory locations within each jail host – using either simple file copy, http fetch, or ftp transfer into the primary host as required.

4.c. Student experiences

To date personal feedback and end of year subject evaluation forms have provided satisfying positive student feedback about the hands on learning approach of RULE. Despite typically having no prior exposure to Unix like operating systems our students respond well to the hands on demystification of Internet service delivery.

Our use of open source software packages allows students to pursue self directed research and experimentation above and beyond the minimum class requirements. By default our FreeBSD jail hosts provide a complete C/C++ development environment, with compiler and all necessary code libraries. Because RULE is accessible from regular Windows hosts over dialup and wireless links, enthusiastic students are able to continue with their experiments from home or other locations around the University campus outside of their 'official' PC laboratory time.

It has been very exciting to see a number of students in each class begin exploring the Unix environments provided by their RULE hosts, experimenting with configurations and software beyond those required by their actual lab classes. A number of students wanted to further explore and expand upon the work being completed during lab hours but were afraid of irreversibly 'breaking' their official RULE host prior to completing their assessment task for the week.

Learning often works best when a student experiments with a software package, breaks it, studies what went wrong and starts again. Due to the ease with which additional RULE hosts can be created on a primary host, we could provide additional temporary hosts to use and experiment with. As noted earlier, it is easy for teaching staff to fix files or entire directory

structures on RULE hosts. When the students get to this point, their temporary host can just be reset to what we define as a default state. They can then start again with new knowledge of what (not) to do.

In addition to the cost savings inherent in re-using existing Windows based lab classes, students find it easier to transition into a Unix command line environment when their first interactions are through familiar Windows based GUI applications. By using PuTTY the FreeBSD command line interface appears within a standard Windows window. Students use the Windows based web browsers they are familiar with to test the servers they have configured. And the WinSCP application allow files to be transferred to and from virtual RULE hosts using a familiar Windows drag and drop paradigm.

5. Protecting the campus from RULE

Whenever students are given free rein of networked machines, network security must be considered. In the case of RULE, security is about protecting the campus network from RULE rather than the other way around. Although students cannot actually get 'raw' access to the underlying network interface (FreeBSD jail hosts force applications to access the network only through Unix `socket()` facilities for conventional TCP or UDP communication), we implemented RULE on an isolated local area network (LAN), connected to the rest of our campus network through an additional FreeBSD based firewall (a regular PC motherboard with two Ethernet interfaces).

This firewall implements rules to match the class projects being run. For some classes, we totally block outbound connections, allowing only inbound connections that originate from elsewhere on the campus network (eg. the Windows PC lab from which the students are doing their lab class). For other classes we may allow outbound connections, but only to specific on campus destinations. We would exclude things like our campus web proxy - the last thing we want are 'interesting' projects on RULE reaching out and annoying people around the Internet.

With FreeBSD's in built traffic shaper, the firewall can also limit traffic speeds in and out of individual jail hosts, providing additional protection to the outside world and ensuring fair use of the network link in and out of the primary host. Our current RULE limits jail hosts to 200 kbit/sec in each direction (the equivalent speed to an entry level broadband Internet connection) - sufficient for the lab classes we have run to date.

Our insistence that students login across a secure shell (ssh) connection ensures that student's communications with their RULE host(s) are

encrypted, including their initial username/password exchange. A feature of ssh known as 'port forwarding' is a potential security hole, allowing TCP connections to bypass the primary host's firewall. For now we impose on students the requirement of responsible use - port forwarding is a conscious act and they will be traced if it abused.

6. RULE virtual host scalability

In the following section we look in detail at the technical issues affecting typical RULE host performance, independent of the teaching and learning considerations discussed so far. These include the impact of CPU speed, available memory (RAM) and hard disk size on the number of jail hosts a single primary host can accommodate. (Readers uninterested with the technical details of motherboard configuration can skip this section without missing key insights into the utility of RULE.)

Our first implementation of RULE in 2003 utilised low power motherboards (500 MHz Intel compatible processors) from VIA Technologies [26], each running five jail hosts. These motherboards were considerably underpowered for computer intensive applications [27], but nevertheless the jail hosts were sufficient for the modest Java development tasks assigned to students that year. In 2004 we moved to a smaller number of substantially more powerful primary hosts – two motherboards based around 2.6 GHz Pentium 4 processors, with 2 Gbyte of RAM each. For the first half of 2004 we configured each primary host with approximately 30 jail hosts. It is worth considering more closely the factors that actually limit the number of jail hosts on a given primary host.

6.a. Practical constraints

Educational networking experiments fall into two broad categories – demonstrating how fast a particular protocol or application functions, and demonstrating correct protocol or application functionality. RULE based around jail hosts is primarily suitable for the latter case, where outright speed of each jail host is not a major requirement. In practice almost all interesting network protocols and networked applications (including mail clients and servers, web servers, DNS servers, etc) can be demonstrated and taught on a live network using the equivalent of a 100 MHz processor. Using this logic, common (circa mid-2004) PC motherboards with CPUs in the 2 GHz to 3 GHz range might be provisioned with at least 20 to 30 concurrent jail hosts.

Another constraint is disk space. A modestly functional FreeBSD jail host requires at least 300 MB of primary host disk space (a standard FreeBSD 4.9 installation takes around 140 MB, but we need to add a few extra packages and allow for temporary directories and some work space). If we expect the

students to do any reasonable work (recompiling open source programs, etc), then 600 MB to 1 GB of disk space per jail host becomes more likely. As an example, our undergraduate lab classes in 2004 were given 600 MB per jail host (leaving 460 MB free for the students) and coped admirably with the tasks described in section 4. At the time of writing, 40 GB to 80 GB IDE hard drives are practically the default for cheap PCs and 250 GB IDE drives are sold in modest to high end systems. Disk drive space appears to be less of a limitation than available CPU speed.

Main memory (RAM) on the primary host provides a further constraint. A minimal FreeBSD system can run with 32 MB of RAM, but at least 100 MB per jail host is more reasonable once a few basic network services are started. Memory consumption is likely to be the primary consideration. At the time of writing it is easy (and cheap) to configure a 2+ GHz CPU motherboard with 512 MB of RAM, but going to 1GB and beyond is non-linearly more expensive (and in any case many standard PC motherboards are limited to a maximum of 4 GB). Assuming 100 MB per jail host, a 2.6 GHz motherboard with 512 MB of RAM, limits the system to 5 jail hosts despite the CPU speed being capable of running 26 hosts. Depending on the relative cost of RAM modules at different sizes, it may be more effective to purchase two motherboards with modest amounts of RAM rather than one motherboard fully loaded.

Finally, physical size and power consumption are key considerations when planning rack space and location. Depending on your particular price point, a collection of small, low power mini-ITX motherboards (e.g. the VIA Technologies range) may be preferable to a single, traditional ATX style motherboard loaded up with a fast CPU and high capacity (read, 'expensive') RAM modules. Although the single ATX style solution might end up with a smaller footprint (because you only need one hard drive and one power supply), the power consumption could easily be higher (and the RAM definitely more expensive) than building with discrete mini-ITX motherboards.

6.b. The impact of RAM

The relationship between primary host RAM and tolerable number of jail hosts is non-linear and highly dependent on the workload amongst the jail hosts. Idle processes from all jail hosts are swapped out of RAM and into the “swap space” (or “virtual RAM”) on the hard drive and loaded back into RAM when they become active again. The overall performance of a RULE system thus critically depends on number of active jail host processes running at any given instant of time.

We ran two simple tests in order to illustrate the performance tradeoffs between primary host RAM and number of jail hosts, using a primary host running on a 2.6 GHz Pentium 4 with 512 MB (and later 1 GB) of RAM.

Our first test simulated a large number of students doing relatively simple activities over an ssh link. The responsiveness per jail host was measured when a 512 MB primary host was configured to have 30 and 100 jail hosts active. Our second test involved repeated http requests to a web server and database backend running on each jail host, measuring the system's response time as the number of jail hosts increased. This was done both for a 512 MB and 1 GB primary host.

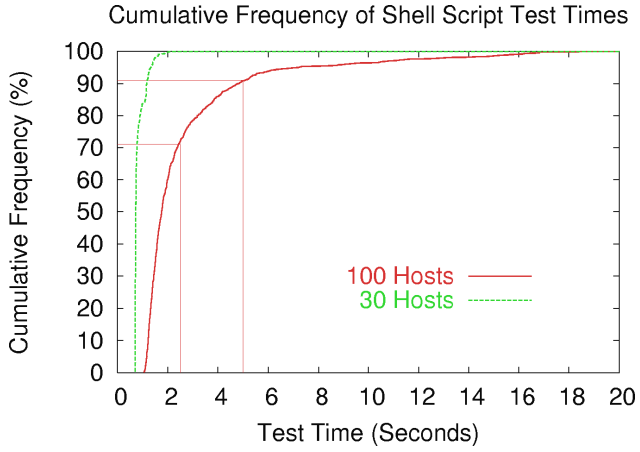

Figure 4: Time taken to complete a 'typical' student exercise with 30 or 100 jail hosts and 512 MB RAM

The first test modeled multiple students concurrently (and repeatedly) performing an introductory Unix exercise involving basic command line shell operations (such as making a directory and editing a file). Figure 4 illustrates that 100% of the simulated exercises completed in less than 2.5 seconds with 30 jail hosts per primary host. With 100 virtual hosts on the primary host, 71% of tests were completed in less than 2.5 seconds, while 9% took more than 5 seconds. In each case the primary host had 512 MB of RAM. The degradation in performance visible in Figure 4 is largely due to each user's jail host processes, competing to be swapped into main memory in a timely fashion.

The second test emulated another typical usage scenario for RULE - teaching users to install and use network application servers. We installed the Apache webserver, MySQL database Server and Client, the “Mod

PHP" Apache module and phpMyAdmin front end for MySQL database administration, on each jail host.

Http was used to request a list of database users from each virtual host and the time taken to load each page was measured. Ten concurrent connections were used and requests were distributed between virtual hosts in a random fashion. The experiment was repeated using between 1 and 45 jail hosts in the pool for the 512MB primary host and 1 to 95 jail hosts in the pool for the 1GB primary host. The average page load time was taken for each pool size and plotted in Figure 5.

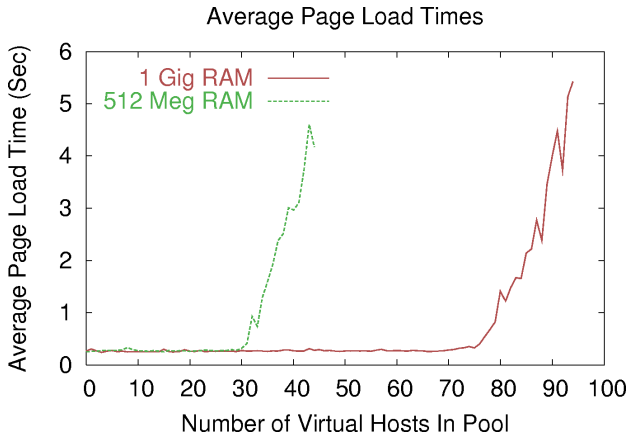

Figure 5: Web transaction times versus number of jail hosts for 512 MB and 1 GB primary hosts

The average page load time is consistently around 0.5 seconds, up to the point where there are too many jail processes trying to be active at the same time. "Too many" is roughly 32 jail hosts in the 512 MB primary host and 77 jail hosts in the 1 GB primary host. Beyond those points, the average page load time skyrockets, due to increased swapping between RAM and hard disk.

The key message from Figure 5 is not the specific point where the performance degrades. Rather, for a given distribution of workload amongst jail hosts, doubling the RAM is going to enable approximately twice the number of jail hosts before performance degrades significantly.

7. Open issues and future directions

RULE is in its early days and we still have a lot to learn and try on all fronts. Resource management within primary hosts is still somewhat crude

and jail hosts do not allow students full control of the host's networking functionality. We are still improving and developing our own internal toolkit to supplement the limited tools provided by FreeBSD itself for managing jails.

Although FreeBSD provides the mechanism for instantiating jails, we still need to provide our own tools for the creation and administration of complete jail hosts. Our solution is called the Jail Host Toolkit (JHT) – a standalone executable that supervises booting, deleting, creating and configuring jail hosts. JHT is GUI application that is used by the primary host's administrator. However, it is also called upon during primary host reboots, to ensure active jail hosts are shutdown and restarted in an orderly fashion.

Resource management between jail hosts is somewhat crude. We isolate disk space consumption by ensuring each jail host exists on a unique disk partition or virtual disk. At this stage we have not developed any RULE specific mechanism for ensuring processes in one jail do not starve processes in another jail of execution time. The primary host treats every process from every jail as essentially equal. Similarly, memory (RAM) consumption is shared across all processes in all jails. For now we are not treating these as critical issues - any student projects that need high performance and predictable computing resources probably shouldn't be deployed on a RULE anyway.

In the longer term we also plan to increase the number of primary hosts and provide students with their own dedicated FreeBSD machines in RULE. This will enable the educational experience to include raw packet access to network interfaces and the ability to control or rebuild the kernels they run. Of course, with complete access to their own primary hosts, it is certain that students will (whether by accident or design) jam their machines and/or scribble all over the hard drive at some point. We are evaluating solutions for remote power cycling/cold rebooting of individual primary hosts and remotely restoring a primary host to pristine, pre-student condition. Ideally the solution will not require staff to physically access the RULE rack space (which would indirectly limit the times of day or week that RULE can be made available to students).

8. Conclusions

It is a continual challenge for educational institutions to keep their students abreast of the latest IP (Internet Protocol) networking services and technologies, while working with scarce laboratory space and limited funding. We have described the development and use of a FreeBSD based Remote Unix Lab Environment (RULE) that provides a modest solution to

our needs. Students 'get their hands dirty' by actually installing and using open source, IP based server, client, and middlebox applications in their own Unix based environments. Not only do our students learn how to use these applications, they are able to learn about modifying and rebuilding the applications.

RULE leverages existing institutional investment in physical infrastructure (such as Windows PCs and campus wide networking) and minimises deployment cost. Our RULE hosts are remotely accessed using secure shell (ssh) from existing PC labs around campus, wireless equipped laptops and dialup hosts (regardless of their chosen operating system at home). This provides flexible opportunities for students to learn and do their project and class work.

To further reduce costs, we utilise FreeBSD's jail functionality to implement jail hosts - virtual Unix hosts that provide each student with their own FreeBSD user space environment to manage and explore. Jail hosts allow our students to act as system administrators within a suitably constrained domain. Jail hosts also allow student lab groups to share a single physical machine, (the primary host) yet run concurrent and distinct instances of applications on the same “well known” port numbers. The administrator of the primary host can effect audits of jail host activities, without requiring cooperation (or knowledge) of the jail host's administrators or users.

There is no specific hardware platform on which RULE is built – it is sufficient that FreeBSD supports a suitable motherboard and that enough resources (disk space, RAM size and CPU speed) are available for the number of jail hosts running on each primary host. In this paper we have provided a rudimentary evaluation of the scaling limits imposed by RAM size, CPU speed and hard drive space. Our first instance of RULE in 2003 was based around three low power (500 MHz/512 MB RAM) motherboards supporting five jail hosts each. In 2004 we migrated to two high end (2.6 GHz/2GB RAM) motherboards to each support 30+ jail hosts. Many aspects of jail host creation and management have been centralised in our Jail Host Toolkit (JHT). RULE is currently deployed under FreeBSD 5.3.

Future versions of RULE will provide students with independent, real Unix hosts. For now, RULE based on virtual hosts has already provided a substantively improved level of hands on and cost effective learning opportunities in our undergraduate (telecommunications engineering) and postgraduate (masters of science in network systems) coursework programs. RULE has also enabled a greater range of networked services (for traffic sources, etc) for postgraduate and post-doctoral research projects in our centre.

Acknowledgments

Clancy Malcolm contributed greatly to the second version of JHT during 2003. Kris Mitchell provided the software development expertise behind our 2004 re-write of JHT.

References

[1] Open Source Initiative. http://www.opensource.org/ [viewed Apr 2005]
[2] The Apache Software Foundation. http://www.apache.org/ [viewed Apr 2005]
[3] Samba. http://www.samba.org/ [viewed Apr 2005]
[4] FreeBSD. http://www.freebsd.org/ [viewed Apr 2005]
[5] H. Kamp, R.N.M. Watson, "Jails: Confining the omnipotent root," SANE2000, 2nd International SANE Conference, May 22-25, 2000, Maastricht, The Netherlands. http://www.nluug.nl/events/sane2000/papers/kamp.pdf
[6] Cygwin Information and Installation. http://www.cygwin.com/ [viewed Apr 2005]
[7] The Linux Home Page at Linux Online. http://www.linux.org/ [viewed Apr 2005]
[8] OpenBSD. http://www.openbsd.org/ [viewed Apr 2005]
[9] NetBSD. http://www.netbsd.org/ [viewed Apr 2005]
[10] M. Aburdene et al, "An Undergraduate Networked Systems laboratory," ACM SIGCOMM, Workshop on Computer Networking, August 2002, Pittsburgh PA.
[11] J. Nieh, C. Vaill, "Supporting a Windows XP/Red Hat Linux dual boot environment," ACM SIGUCCS, 2003, San Antonio TX, USA.
[12] OpenSSH. http://www.openssh.org/ [viewed Apr 2005]
[13] VMware: Enterprise-Class Virtualization Software. http://www.vmware.com/ [viewed Apr 2005]
[14] bochs: The Open Source IA-32 Emulation Project. http://bochs.sourceforge.net/ [viewed Apr 2005]
[15] Kevin Lawton, "Plex86 x86 Virtual Machine Project," http://plex86.sourceforge.net/ [viewed May 2003]
[16] QEMU. http://fabrice.bellard.free.fr/qemu/ [viewed Apr 2005]
[17] J. Nieh, C. Vaill, "Experiences teaching operating systems using virtual platforms and linux," ACM SIGCSE Bulletin, 2005.
[18] Simon Tatham, "PuTTY: A Free Win32 Telnet/SSH Client." http://www.chiark.greenend.org.uk/~sgtatham/putty/ [viewed Apr 2004]
[19] Martin Prikryl, "WinSCP," http://winscp.sourceforge.net/eng/ [viewed Apr 2005]
[20] thttpd HTTP server. http://www.acme.com/software/thttpd/ [viewed Apr 2005]
[21] R. Kaes, S. Young, "tinyproxy". http://tinyproxy.sourceforge.net/ [viewed Apr 2005]
[22] C. Lee, "DNS". http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/network-dns.html [viewed Apr 2005]
[23] PHP: Hypertext Preprocessor. http://www.php.net/ [viewed Apr 2005]
[24] MySQL. http://www.mysql.com/ [viewed Apr 2005]
[25] Rsync. http://samba.anu.edu.au/rsync/ [viewed Apr 2005]
[26] Via Technologies Inc. http://www.viavpsd.com/ [viewed Apr 2005]
[27] [27] Epia M10000 Review. http://www.epiacenter.com/modules.php?name=Content&pa=showpage&pid=21

Appendix A: Limitations of jail host environments

A jail host supports all conventional, kernel-mediated TCP and UDP based communication. It runs its own instances of the ssh daemon, has its own password files, and has its own copy of the FreeBSD file system. The primary host has multiple IP addresses - one for itself and one for each jail host.

User-space applications you can run on a regular FreeBSD machine will generally run unchanged inside an equivalent jail host.

Restrictions are enforced by the primary host's kernel to ensure jail host processes never see (and cannot access) other processes and files outside their constrained context. Primary host processes have no such restrictions.

A jail host does not completely replicate the environment of a regular FreeBSD host. The limitations primarily relate to the jail host's networking and kernel functionality. In FreeBSD 4.x these include:

- Networking
- A jail host has a single network interface and a single IP address
- A jail host cannot get raw access to the network interface (e.g. for network sniffing with tcpdump, building custom UDP frames for traceroute, sending ICMP packets to other hosts for ping, etc....)
- Jail hosts are currently IPv4-only
- Kernel
- File systems cannot be mounted or unmounted from within the jail host
- The kernel, and kernel system variables, cannot be modified from within a jail host
- Access to physical devices is substantially constrained

Some applications need fine tuning to handle the fact that localhost (traditionally 127.0.0.1) is silently mapped to the jail host's actual IP address. For example, the ssh daemon within each jail host must be told not to use “localhost” when setting up X11 forwarding back across inbound ssh connections (by adding the line “`X11UseLocalhost no`” to /etc/ssh/sshd_config.)

Other resources that need to be re-configured in the primary and jail hosts include the default set of ptys – virtual terminals that support ssh logins to each jail host. The default FreeBSD configuration has /dev/ptyp0 through /dev/ptypv defined. With many jail hosts, and multiple ssh logins per jail host, the system can quickly run out of spare ptys for new logins. The solution is to run “`./MAKEDEV pty1 pty2 pty3`” in the /dev/ directories of the primary host and each jail host.

More details about the restrictions imposed by jails can be found in Kamp and Watson's SANE2000 paper [5].

Appendix B: Glossary

DNS	Domain Name System - A system fundamental to the Internet by which Internet domain names (eg. www.swin.edu.au) are mapped to IP addresses.
FreeBSD	A free, open source, Unix-like operating system.
Host	A entity or node on a network.
HTTP	Hyper Text Transfer Trotocol - The protocol by which hyper text (web pages) are transferred across an IP network.
I/O ports	Input/Output ports – The physical ports of a computer used to interconnect with hardware devices.
IP	Internet Protocol - The protocol standard on which the Internet is built. IP address - Internet Protocol Address - (often presented in the form "136.186.1.35").
Jail	A FreeBSD concept that allows applications to be contained within a constrained environment.
Kernel	The core component of an operating system that mediates user space applications access to hardware.
Linux	A free, open source, Unix like operating system.
Motherboard	The main circuit board of a computer.
NetBSD	A free, open source, Unix like operating system.
Open source	A software package where the programming source code is freely available for inspection and modification.
OpenBSD	A free, open source, Unix like operating system.
Primary host	A machine that supports multiple virtual hosts upon it.
RULE	Remote Unix Lab Environment - A networked server allowing multiple users to login to their own complete FreeBSD virtual hosts.
SSH	Secure Shell - The most common means of secure login to Unix like operating systems.
User space	The normal operating environment for applications, its use is mediated by the kernel.
Virtual host	A virtual FreeBSD host that runs on a primary host (in user space), but looks for all purposes like a regular machine
Virtual machine	A virtual computer being fully emulated at the hardware level on another computer (and thus has more overheads than a virtual host).

Grenville Armitage and Warren Harrop
Centre for Advanced Internet Architectures
Swinburne University of Technology
PO Box 218, Hawthorn, Victoria 3122
Email: garmitage@swin.edu.au, wazz@bud.cc.swin.edu.au
Web: http://www.caia.swin.edu.au/