A Note on Building an Internet Laboratory

Michael S. Kester
Dept. of Computer Science
Columbia University
msk2117@columbia.edu

Gil Zussman
Dept. of Electrical Engineering
Columbia University
gil@ee.columbia.edu

***Abstract*—The Networking Lab class, offered at Columbia University, follows the series of detailed laboratory experiments presented in *Mastering Networks: An Internet Lab Manual* by Jörg Liebeherr and Magda El Zarki [8]. In order to conduct these experiments, there is a need to construct and maintain an Internet laboratory composed of routers, hubs, and Linux machines. Since the students performing the experiments have root access to the machines, we have encountered many misconfigurations that affect the learning experience. Therefore, we have developed an efficient process that allows setting up the lab quickly and that also supports quick recovery from misconfigurations. In this note, we present the process and the lessons learned during its development.**

***Index Terms*—Internet laboratory, Mastering networks, deployment, maintenance, teaching, experiments.**

I. Introduction

The Networking Lab class focuses on applying networking concepts in a real network. It follows a series of detailed laboratory experiments presented in *Mastering Networks: An Internet Lab Manual* by Jörg Liebeherr and Magda El Zarki [8]. These experiments provide students hands-on experience that allows them to gain a good understanding of some common network protocols operation and design principles. In order to conduct these experiments, there is a need to construct and maintain an Internet laboratory composed of routers, hubs, and Linux machines.

Since the students performing the experiments need root access to the machines and the machines configurations change throughout the semester, we found that the lab construction process has to be repeated at the beginning of each semester. Moreover, we have encountered many misconfigurations that cannot be easily fixed during a lab session, thereby affecting the students' learning experience. There are a couple of alternatives for resolving these issues, including using virtual machines (VMs) [9] or live CDs [7]. We developed a third method that allows quick lab setup and also supports quick recovery from misconfigurations. This note briefly outlines the process and the lessons learned during its development.

II. Lab Construction - Motivation & Objective

At the beginning of each semester, the teaching staff is required to prepare the networking lab machines for the incoming class.[1] During the semester, students need to have root access in order to be able to exercise the functionality of real systems as described in the manual. Unfortunately, giving root access to students tends to lead to misconfigurations. Hence, in addition to the basic configuration, the setup process has been designed such that students will not need to struggle with misconfigured or unstable machines.

There are several options for configuring the lab in a way which attempts to avoid major machine misconfigurations by the students. In past semesters, we used *VMs* running on top of the physical hardware. This in practice however was suboptimal. The VMs did not always run identically to the way they would run on bare hardware. Thus, when an exercise was not working, students had difficulty determining if their configuration was to blame or the VM itself. The students also reported that the VMs were slow, unnecessarily lengthening the experiments.

Working from a *live CD* would also protect against accidental misconfiguration. However, there are three potential drawbacks to using a CD: (i) Non-default packages must be installed on each reboot. (ii) Students are unable to save a configuration and return to it. (iii) Working from a live CD has performance implications. Liebeherr and El Zarki solve the first issue by providing a custom live CD on their website [7], [8]. The second problem is only a concern for the longer labs; those that are too long to complete in one sitting but not so long that they need to be broken across assignments. The final point can be overcome by using the copy2ram or a similar flag when booting but this too forces a wait as each system must load the full OS to RAM at each boot; on our hardware this is a non-trivial period. The alternative of working from the CD itself is even worse as the system must read from the CD to perform even the most basic commands.

We chose to implement another option in which we *multi-boot the machines using identical partitions*. If the students break something fundamental, a single change to `menu.lst` gives them a brand new machine. This saves the students and the teaching staff from a potentially lengthy troubleshooting exercise. Additionally, this approach allows separate sections to avoid the problems created by another, and prevents a later section from arriving at a machine that is already fully configured.

Below, we describe the method we used to build such a machine and to clone it to the rest of the lab. From our experience, the overall process takes about 10 to 20 person-hours (additional time should be reserved for complications which may arise). If starting from scratch (i.e., having to

[1] Most of this discussion is equally applicable to a rebuild performed each semester as it is to building a new lab.

physically install the equipment and build the base image), 40 hours might be more typical.

III. The Build

We use a simple method to roll out the new build which relies only on basic Linux tools. We first section the disk into multiple partitions. In our case we used only two, but given a larger disk it might be advantageous to build four or more. We intended the second partition to be used only for fail-over in the event of a major problem during a lab session. However, if desired, the relatively small footprint of the Linux operating system could allow each student to be responsible for a set machine and partition. We first build the machine on a single partition, then use `dd` to copy it to the other partitions, and finally use `dd` and `netcat` to roll the image out to all of the other machines.

A. Build the Base Machine

Building the machine is fairly straightforward. We choose a desktop distribution. Though many of the packages are superfluous, this allows students with less command line experience the comfort of a GUI. The book is written for use with Red Hat 9.0 [5] or more recent. We use Fedora 13 [2], the desktop fork of Red Hat, with little need for annotating the book instructions. Other Red Hat derivatives like Mandriva or CentOS would also likely work.

It is wise to keep the lab routers away from the general campus infrastructure, thus, there are no Internet connections in our lab. Therefore, either physical media or a temporary connection is required. We believe that it is best for students to use a modern and updated system. Hence, we suggest doing a standard live CD install followed by an update. In our lab we were able to use a netbook connected to the campus wireless network as a gateway router. Next, we turn our attention to the build itself.

Our machines have a single logical 36GB drive. We partitioned ours into two separate 15GB partitions for root and the rest as swap. These are fairly generous partition sizes. We could probably use 8GB root partitions and fit four on the drives we have. However the disk is partitioned, it is important that each of the other partitions is the same size or larger than the original build partition or the clone will fail. Each of the machines can share a single swap partition, and therefore, we include only one. We do a basic installation of Fedora 13, which was current at the time of this writing. During the installation there are options for including or removing certain packages. Since we use these machines on an isolated network, we can safely exclude a number of the suggested packages. We remove games, email, and any sort of Internet only package, for example chat clients. We also add Wireshark and bind, which are definitely needed. Further, we added some additional packages that are needed, or we thought may be needed: brctl, dhcp, gbrctl, mrouted, mtools, named, nmap-frontend, openssh, openswan, quagga, telnet-server, tftp, tftp-server, ttcp, and xinetd. Once all the packages are installed, updated, and a basic account is set up, we move on to the cloning.

B. Copy the Build to the Remaining Partitions

We do a bit for bit copy of our build machine to the other partitions together with a little clean up so that each of the partitions work properly.

First we need to remove the persistent Ethernet names:

```
rm /etc/udev/rules.d/70-persistent-net.rules
```

This file is generally generated on each reboot, if it does not exist. The purpose of this file is to ensure that `udev` chooses the same name for a specific adapter across reboots. If we copy this file during the cloning process, `udev` will see the Ethernet cards on the clone as new hardware. This causes connectivity issues. By removing the file now, before we clone the build, we save ourselves from having to do delete the file on each partition of each machine.

To ensure that we achieve a clean copy, we boot the machine using a live CD. Here we can use just about any distribution that provides a live CD as we rely only on `coreutils` and `BASH`. We run:

```
dd if=/dev/sda1 | tee >(dd of=/dev/sda2) | \
tee >(dd of=/dev/sda3)| dd of=/dev/sda4
```

where 'sda1' is the partition that we built our machine on, and 'sda2' to 'sda4' are the empty partitions we want to clone our drive to. These labels may vary depending on the hardware used. In our case we had two partitions, and therefore, we use only the first and last commands from this pipeline.

Once the system is copied, we need to make a couple changes. We let the first partition remain the boot partition. Fedora 13 uses legacy grub, and therefore, we only need to worry about modifying */boot/grub/menu.lst* on the boot partition, generally the first partition. The remaining partitions will just have unused menu.lst files.

We need to create and set a new UUID on the copied partition(s) and add the appropriate lines to menu.lst. First, we generate a UUID with `uuidgen -t` and then assign the new UUID to the copied partition using `tune2fs -U /dev/sda2`. This is repeated as necessary such that each partition has a new UUID. Now that referable IDs have been created, we copy one entry of menu.lst for each new partition and change the UUID accordingly. It is also advisable to label the title build1, build2, etc. such that each may be easily identified from the grub menu at boot.

At this point, the system is able to boot the other partitions, but only the first partition will be mounted. We need to update fstab on each new partition. We can do this from any of the boot systems by simply mounting each partition and changing */etc/fstab* to reflect the corresponding UUID.

Additionally, it is prudent to set grub timeout to 0, and hide the menu so that the students are less likely to change something they should not change. If we were assigning responsibility for certain machines to specific students, we would want to expand the labeling and increase the time-out. As described, booting to a clean machine requires only a small change in menu.lst, namely the "default" value.

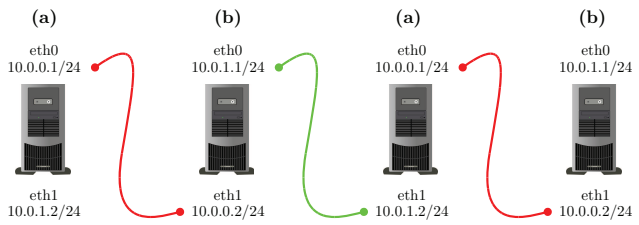

Fig. 1. Chaining the PCs together.

C. Copy the Disk Image to Additional Machines

Next, we clone our build to the other physical machines. We use `dd` and `netcat` in a similar pipeline to the one previously mentioned. Again, we will need to boot from a live CD. The only requirement for this CD is that the distribution includes both utilities. Also, to avoid burning several CDs, we choose one that boots to RAM.

We used the live distribution of Fedora 13 as well as the Debian derivative Knoppix [3]. In Fedora, press any key at the count down. At the grub menu highlight "boot" and press `Tab`. Replace 'liveimg' with 'live_ram' and press `Enter`. In Knoppix, we use 'knoppix toram 2' which boots Knoppix to RAM at run level 2. Since we only need the command line, there is no need to load the desktop environment.

We then push the build from one machine to the next in a single large daisy chain. Since each machine has two Ethernet ports we configure them as either:

(a) eth0 10.0.0.1/24
    eth1 10.0.1.2/24
(b) eth0 10.0.1.1/24
    eth1 10.0.0.2/24

Physically we wire each *eth0* to the next machine's *eth1*, alternating Ethernet configurations between (a) and (b). One station is shown in Fig. 1, but stations can also be chained one to the next. All but the first and last machines will have both Ethernet ports connected and each machine can ping the next in line through eth0.

Now starting from the *last* machine in the chain we set up a listener and a writer:

```
nc -l -p 10002 | dd of=/dev/sda
```

Then working backward along the chain, we alternate between:

(a) nc -l -p 10012 | of=/dev/sda | nc 10.0.0.2 10002
(b) nc -l -p 10002 | dd of=/dev/sda | nc 10.0.1.2 10012

Finally, having worked all the way back to the first machine in the chain, we start the copy with:

```
dd if=/dev/sda | nc -q 3 10.0.0.2 10002
```

Here *sda* is the entire disk, not just a single partition. As mentioned previously, the label *sda* may change depending on hardware. Also note that the syntax of `nc` is a little different depending on the distribution chosen. Netcat generally closes the socket once the input stream closes. Namely, when `dd` finishes reading the entire disk on the first machine. This is true in the live CD version of Fedora 13. However, to achieve the same behavior in Knoppix it was necessary to use the `-q N` flag which is supposed to close the socket 'N' seconds after the input stream reads `EOF`. We found that while Knoppix version does close the socket when using the 'q' flag, it does not wait *N* seconds but rather quits immediately which for our purposes is adequate.

Given that a full disk copy across a 100Mbs link will take some time, it is useful to setup and test a small file through the chain before running the entire disk. We advise against using Fedora, as it takes some time to load to RAM. Using Knoppix we could chain all 20 computers in our environment relatively quickly. Damn Small Linux [1] would be another option worth trying, as would any of the other minimal distributions.

Now that the systems are rolled out we just need to change the name in a couple places. Since the machine was built as PC1 (or some other static name), we need to change it in each of the clones. The name needs to be updated in two places:

```
/etc/hosts
/etc/sysconfig/network
```

Once this is complete, repeat the renaming on each partition. Each machine should now function as though it was built individually, and each duplicate partition can be used as a fresh machine if one becomes compromised.

D. Final Steps

Now that the systems are built, there are a few configuration settings to check. Liebeherr and El Zarki provide scripts on their site [8] to help automate the process. The server configurations could also be done prior to cloning. In our case, the TAs did the following four steps:

1) Enable telnet, tftp, and named within xinetd.
2) Disable SELinux.
3) Disable iptables.
4) Reset the routers to factory defaults.

These four steps could be done as a walk-through given to the students to expose them to Linux features falling outside the scope of this course, or the scripts can be run by the TAs before the first students arrive. To enable telnet, tftp, and named, run the following:

```
chkconfig telnet on
chkconfig tftp on
chkconfig named on
```

Similarly, one can disable iptables with

```
chkconfig iptables off
```

SELinux has a configuration file located at /etc/selinux/config which should be edited to include this line:

```
SELINUX=disabled
```

Additionally, ensure the file "/etc/xinetd.d/telnet" includes the line:

```
disable = no
```

Following these changes, each machine should be rebooted.

Finally, the Cisco routers should be reset to factory defaults. Since this involves Cisco commands that the students have yet

to learn, we suggest the TAs reset the routers at the same time as the server builds. This snippet can be found on Cisco's website [6].

1) router# configure terminal
   router(config)# config-register 0x2102
   router(config)# end
2) router# write erase
3) router# reload
   System configuration has been modified. Save? [yes/no]: n
   Proceed with reload? [confirm]

Upon reboot, a brief configuration dialog must be completed after which the router is ready for use.

IV. Conclusions

We presented a detailed method for building and configuring the laboratory associated with a hands-on introductory course to networking. From our experience, this method creates a setup with quicker response time than provided by other methods (virtual machines and live CDs). Moreover, it lets the students focus on the lab experiments rather than on fixing misconfigurations that may have been caused by other students. Hence, the proposed method improves the overall learning experience.

Acknowledgments

We thank Tarun Sharma for his help with implementing and testing the process. This work was supported in part by NSF grant CNS-10-54856 and CIAN NSF ERC under grant EEC-0812072.

References

[1] DSL information. http://www.damnsmalllinux.org/, Nov. 2011.
[2] Fedora project homepage. http://fedoraproject.org, Nov. 2011.
[3] KNOPPIX - Live Linux filesystem on CD. http://www.knopper.net/knoppix/index-en.html, Nov. 2011.
[4] netem — The Linux Foundation. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem, Nov. 2011.
[5] redhat.com — The world's open source leader. http://www.redhat.com, Nov. 2011.
[6] Reset a Cisco router to factory default settings - Cisco Systems. http://www.cisco.com/en/US/products/sw/iosswrel/ps5187/products_tech_note09186a00802017a1.shtml, Nov. 2011.
[7] J. Liebeherr. TCPIP-LAB.NET. http://www.tcpip-lab.net/, Nov. 2011.
[8] J. Liebeherr and M. El Zarki, *Mastering networks: an Internet lab manual*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.
[9] J. Nieh and C. Vaill, "Experiences teaching operating systems using virtual platforms and Linux," *ACM SIGCSE Bull.*, vol. 37, pp. 520–524, Feb. 2005.